Titre: Title:	Synchronisation de traces distribuées à l'aide d'événements de bas niveau
Auteur: Author:	Benjamin Poirier
Date:	2010
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Poirier, B. (2010). Synchronisation de traces distribuées à l'aide d'événements de bas niveau [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/261/

UNIVERSITÉ DE MONTRÉAL

SYNCHRONISATION DE TRACES DISTRIBUÉES À L'AIDE D'ÉVÉNEMENTS DE BAS NIVEAU

BENJAMIN POIRIER
DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION DU DIPLÔME DE
MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
AVRIL 2010

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

SYNCHRONISATION DE TRACES DISTRIBUÉES À L'AIDE D'ÉVÉNEMENTS DE BAS NIVEAU

présenté par : POIRIER, Benjamin
en vue de l'obtention du diplôme de : Maîtrise ès Sciences Appliquées
a été dûment accepté par le jury constitué de :

Mme. BOUCHENEB, Hanifa, Doctorat, présidente.
M. ROY, Robert, Ph.D., membre et directeur de recherche.
M. DAGENAIS, Michel, Ph.D., membre et codirecteur de recherche.
M. BOYER, François-Raymond, Ph.D., membre.

REMERCIEMENTS

Je tiens à remercier mes directeur et codirecteur de recherche, Robert Roy et Michel Dagenais, pour leur attention au détail et leur disponibilité. Je remercie le personnel technique du département de génie informatique et génie logiciel, en particulier Francis Gagnon et Jean-Marc Chevalier, pour leur appui répété lors de l'enregistrement de traces sur les ordinateurs du département. Je remercie également mes collègues du laboratoire DORSAL, en particulier Pierre-Marc Fournier et Mathieu Desnoyers, d'avoir partagé avec moi leurs connaissances techniques approfondies.

Finalement, je tiens à reconnaître le soutien financier offert par le Conseil de recherche en sciences naturelles et en génie du Canada (CRSNG) ainsi que la contribution d'Ericsson Software Research et de Recherche & développement pour la défense Canada (DRDC).

RÉSUMÉ

Le traçage d'événement a prouvé son efficacité dans l'identification de problèmes de fonctionnalité et de performance. Il peut s'appliquer dans le développement d'applications, de systèmes d'exploitation et de pilotes matériels. Pour l'appliquer également à des systèmes distribués, il est possible d'enregistrer des traces individuellement sur chaque noeud puis de regrouper ces traces lors de l'analyse a posteriori. Il faut alors que les estampilles de temps associées aux événements soient synchronisées avec fidélité et justesse.

Ce mémoire traite de la synchronisation hors ligne de groupes de traces enregistrées sur un système distribué. L'objectif est d'identifier un algorithme de synchronisation de trace précis, qui s'exécute en temps linéaire par rapport au nombre d'événements dans le groupe de traces, qui garantit l'absence d'inversion de message et qui peut déterminer des bornes sur la justesse.

Dans l'approche préconisée, le traçage noyau enregistre avec une faible intrusivité des événements correspondant à l'envoi et la réception de messages réseau. Les relations d'ordre entre ces événements sont utilisées pour construire une base de temps globale. L'approche se concentre sur la synchronisation de paires de traces à l'aide de fonctions linéaires de correction d'horloge. Les paramètres de ces fonctions sont identifiés à l'aide de l'algorithme basé sur la méthode des enveloppes convexes. Cet algorithme garantit l'absence d'inversion de message et il a été étendu afin de déterminer des bornes sur la justesse à tout moment lors du traçage.

L'algorithme a été intégré à un outil de visualisation de trace. Son application à des traces réelles, variées et de grandes tailles révèle que la précision de la synchronisation est favorisée par l'utilisation d'un réseau à plus faible latence ou d'un débit de message plus élevé. En contrepartie, une durée de traçage plus longue réduit la précision et fausse la détermination des bornes de justesse. Cette situation est mise en évidence grâce à l'utilisation de métriques sur les temps de transfert des messages après synchronisation. L'étude de la performance de l'algorithme confirme qu'il peut réaliser la synchronisation avec un ordre d'exécution linéaire. La détermination des bornes de justesse a un comportement quadratique dans le pire des cas, mais, avec des traces réelles, elle s'exécute en temps presque linéaire. Lors de nos expériences, nous avons atteint une justesse de ± 15 us et une précision estimée à 9 us sur un réseau comportant une latence minimale estimée à 39 us.

La détermination de bornes sur la justesse de la synchronisation représente une avancée scientifique alors que l'implantation efficace de l'algorithme de synchronisation dans un système de traçage pratique représente une avancée technique.

ABSTRACT

Event tracing has proven to be a valuable tool for identifying functional and performance problems. It has helped to identify problems at the application, operating system and device driver level. In order to extend its benefits to distributed systems, one approach is to record traces individually on each node and to analyze them in a post-processing step. In order for this to be meaningful, the timestamps in the traces have to be synchronized with precision and accuracy.

This dissertation focuses on offline synchronization of traces recorded on distributed systems. The objective is to identify a trace synchronization algorithm that is precise, has a linear run time order in regards to the number of events in the traces, that can guarantee the absence of message inversions and identify accuracy bounds.

The method put forward uses kernel tracing to record network events with a low intrusiveness. A global timebase is built by analyzing the strict ordering relationships between events that correspond to the emission and reception of messages. We concentrate on the synchronization of pair of traces using linear clock correction functions. The parameters of these functions are identified using the convex hull algorithm. It guarantees the absence of message inversions and it has been extended to identify accuracy bounds at any point in the tracing interval.

This algorithm was contributed to a trace analysis tool and was used on a variety of long running and large traces recorded on real systems. The experiments conducted show that offline synchronization accuracy is improved by using a network with lower latency and by using a higher message rate. With a constant message rate, lengthening the trace duration reduces precision and gives a false impression of improving accuracy. This is detected using metrics based on message propagation delays after synchronization. Synchronization factors can be found in linear time. The time to find accuracy bounds is quadratic in the worst case but it scales almost linearly on practical traces. During our experiments, we have achieved a synchronization accuracy of ± 15 us and an estimated precision of 9 us on a network with an estimated minimum propagation delay of 39 us.

Identifying strict accuracy bounds during offline synchronization of traces represents a scientific advance whereas the efficient implementation of a trace synchronization algorithm in a practical tool represents a technical progress.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

GPS	Global Positioning System
GSM	Global System for Mobile communications
KTAU	Kernel Tuning and Analysis Utilities
LTTng	Linux Trace Toolkit Next Generation
MPE	Multi-Processing Environment
MPI	Message Passing Interface
NTP	Network Time Protocol
OpenMP	Open Multi-Processing
PLL	Phase Lock Loop
PTP	Precision Time Protocol
SA	Sample After
SB	Sample Before
SBA	Sample Before and After
TAU	Tuning and Analysis Utilities
TCP	Transmission Control Protocol

CHAPITRE 1

INTRODUCTION

1.1 Problème étudié et buts poursuivis

Que ce soit pour identifier des problèmes de performance ou de fonctionnalité, le traçage a démontré son utilité. Il est utilisé en recherche et en industrie. Il a permis d'identifier des problèmes mettant en jeu applications, pilotes matériel et systèmes d'exploitation.

Les traceurs sont des outils de développement qui enregistrent des événements durant l'exécution d'un système. Un événement est constitué d'un identificateur, d'une estampille temporelle et, optionnellement, de paramètres fournissant des informations supplémentaires. Les événements sont générés lors de l'exécution d'appels de fonction particuliers placés à des endroits stratégiques dans le code d'un programme. L'analyse des événements enregistrés dans les traces permet de déduire l'état du système et de mesurer sa performance.

Certains problèmes mettent en jeu des systèmes composés de noeuds distincts qui interagissent par un réseau. C'est le cas des architectures client-serveur ou de calcul distribué. Pour identifier ces problèmes, il est nécessaire d'avoir une compréhension globale de l'exécution. Afin de bénéficier du traçage sur de tels systèmes, il est possible d'enregister des traces individuellement sur chaque noeud. Chacune de ces traces est prise sur un système qui comporte une horloge locale. Or, il y a le plus souvent absence d'horloge globale. Il faut donc effectuer une synchronisation des estampilles temporelles des événements afin de reconstruire une base de temps globale. La synchronisation de traces d'exécution enregistrées sur des systèmes répartis constitue l'intérêt du présent mémoire.

Quels sont les buts poursuivis ? Un système de traçage est un instrument de mesure ayant une certaine qualité métrologique. Afin de présenter des résultats exacts, il se doit d'avoir deux propriétés :

1. Être fidèle. Il est important de ne pas modifier ce que l'on tente de mesurer. L'exécution du traceur doit être aussi peu intrusive que possible pour que les estampilles temporelles soient précises et que le comportement du système soit presque le même qu'en l'absence de traçage. De même, la différence entre les estampilles temporelles de deux événements concurrents doit être aussi faible que possible après correction par un algorithme de synchronisation.
2. Être juste. Lorsque l'on tente d'identifier un problème, le comportement du système est mis en doute. On ne peut donc présenter de l'information qui soit elle aussi possiblement

erronée. Il doit être possible de garantir que l'ordre des événements dans des traces synchronisées corresponde à leur ordre réel (ou du moins qu'il n'y ait pas de violation d'ordre partiel).

Afin de réduire l'intrusivité du traceur, les traces sont généralement sauvegardées durant l'exécution, puis analysées a posteriori. Ceci implique que le volume de données à analyser croît avec le débit d'événements et la durée de la trace. Il faut donc un système qui puisse traiter efficacement de grandes traces et qui se comporte de façon gracieuse par rapport à une augmentation de la quantité d'événements.

Lorsque des traces distribuées sont analysées conjointement, il peut se produire qu'un message semble avoir été reçu avant même d'avoir été envoyé. Cette situation fâcheuse est désignée comme une « inversion de message. » Cela est possible si la latence de communication est plus petite que la justesse de la synchronisation.

L'objectif de la présente recherche est de trouver un algorithme de synchronisation de trace précis, qui s'exécute en temps linéaire par rapport au nombre d'événements dans un groupe de traces, qui garantit l'absence d'inversion de messages et qui peut déterminer des bornes sur la justesse.

1.2 Démarche de l'ensemble du travail

L'approche envisagée est de réaliser la synchronisation a posteriori. Ceci est fait en se fiant sur des relations d'ordre partiel entre des événements se produisant dans des traces enregistrées avec des horloges différentes. Ces relations permettent d'établir des fonctions de conversion entre ces horloges. Cette recherche se concentre plus particulièrement sur :

- l'utilisation d'événements d'envoi et de réception de paquets réseau enregistrés à un bas niveau dans le système d'exploitation
- la synchronisation d'une paire de noeuds
- l'utilisation de fonctions de conversion linéaires

Nous avons dans un premier temps évalué les algorithmes disponibles pour la synchronisation de traces par rapport aux fonctionnalités et aux propriétés énoncées à la section 1.1. Dans un second temps, nous avons conçu une amélioration à un de ces algorithmes afin d'en mesurer la justesse.

Nous nous sommes concentrés sur l'algorithme basé sur la méthode des enveloppes convexes. À l'aide de cet algorithme, nous sommes capables de synchroniser une paire de noeuds en temps linéaire sans inversion de message. Nous allons décrire une nouvelle extension à cet algorithme permettant de calculer une borne sur la justesse et donc de garantir une certaine précision.

1.3 Organisation générale du document

Le chapitre 2 présente une revue de la littérature concernant la synchronisation de temps, en particulier les algorithmes applicables à la synchronisation de traces a posteriori. Le chapitre 3 intègre le contenu de l'article « Accurate Offline Synchronization of Distributed Traces Using Kernel-level Events. » Cet article présente l'essentiel des développements réalisés durant la recherche. Il inclut une description de l'algorithme permettant de mesurer la justesse de la synchronisation ainsi qu'une description de la méthode nécessaire pour enregistrer avec une faible intrusivité les événements réseau nécessaires à la synchronisation. L'algorithme présenté répond aux objectifs énoncés à la section 1.1. Suivent les résultats et l'analyse d'expériences impliquant l'utilisation de ce système de traçage dans diverses conditions. L'influence sur la précision et la justesse de paramètres tels que le type de réseau et la durée de traçage y est étudiée. Le chapitre 4 inclut une discussion complémentaire à celle présentée dans l'article. Cette discussion met de l'avant certains résultats inattendus et compare les résultats obtenus avec ceux présentés dans la littérature. Le chapitre 5 conclut le mémoire par une synthèse critique ainsi que quelques pistes de recherche future.

CHAPITRE 2

REVUE DE LITTÉRATURE

Ce chapitre présente l'évolution jusqu'à l'état de l'art du domaine de la synchronisation comme étape préalable à l'analyse de performance, en utilisant des traces d'événements provenant de systèmes distribués. Ceci comprend les premiers algorithmes d'ordre partiel et leurs limites, puis les équations et hypothèses fondatrices d'une seconde classe d'algorithmes de synchronisation en temps absolu. Sont alors présentés les algorithmes de synchronisation en ligne et les algorithmes hors ligne. Une attention particulière est portée à ces derniers qui sont le sujet d'intérêt du présent mémoire.

Cette revue permet de se familiariser avec le domaine : ses fondements théoriques, son évolution et ses besoins actuels. L'objectif est double : d'une part, évaluer différents algorithmes selon leur aptitude à synchroniser des traces d'événements ; d'autre part, de pouvoir comprendre l'origine et évaluer la création d'un nouvel algorithme de synchronisation, ce qui sera le sujet du prochain chapitre.

2.1 Temps logique

Une première approche pour la synchronisation de groupes de traces est d'utiliser des paires d'événements qui ont un lien tel que l'un des deux se produit nécessairement avant l'autre. En identifiant plusieurs de ces liens dans des traces différentes, il est possible d'établir un ordre partiel entre les événements.

Le travail fondateur pour cette approche a été effectué par Lamport qui a formalisé le concept d'ordre partiel sur les événements (Lamport, 1978). Il a aussi proposé une façon de l'utiliser pour confirmer des relations de causalité : les horloges logiques. Une extension de cette approche, les horloges vectorielles, permet de déduire des relations de causalité à partir de l'ordre partiel (la relation opposée) mais comporte un coût en mémoire croissant en fonction du nombre de noeuds. Un raffinement subséquent, les horloges en arbres (Landes, 2007), réduit ce coût dans certaines situations. Ces algorithmes peuvent être utilisés pour synchroniser des traces d'événements selon un temps logique, c'est-à-dire de façon à respecter les relations d'ordre partiel sans égard au temps réel.

La plupart des chercheurs s'entendent sur le fait qu'une synchronisation par rapport à un temps logique est suffisante pour atteindre le premier objectif d'un système de traçage distribué, l'identification de problèmes fonctionnels, mais insuffisante dans le cas du deuxième

(Hofmann et Hilgers, 1998; Doleschal *et al.*, 2008) ; l'identification de problèmes de performance nécessite des mesures par rapport au temps « absolu » ou « physique » et nécessite donc une autre classe d'algorithmes.

2.2 Temps absolu

Dans le cadre du traçage, la synchronisation en temps absolu a pour but d'évaluer et de corriger la différence entre les horloges de chaque noeud du système tracé. Ceci permet de passer d'une base de temps locale différente pour chaque noeud à une base de temps globale. Le but est de présenter l'ensemble des événements tracés sur une base de temps homogène. Pour ce faire, deux approches sont envisageables : la synchronisation des horloges durant le traçage (désignée par la suite comme étant « en ligne ») ou la synchronisation des traces durant une étape de traitement ultérieure (désignée par la suite comme étant « hors ligne »). Dans un cas comme dans l'autre, la tâche est complexifiée par l'inexactitude des horloges locales et l'incertitude des délais de communication entre les noeuds tracés.

2.2.1 Modèle d'horloge

Une étape préliminaire nécessaire à l'élaboration et la compréhension des algorithmes de synchronisation en temps absolu est la modélisation de l'horloge locale de chaque noeud. Une telle modélisation sert à établir les hypothèses de départ d'un algorithme. Un article sur le comportement des oscillateurs et leurs sources d'erreurs (Ellingson et Kulpinski, 1973) a établi le modèle de référence pour plusieurs algorithmes de synchronisation (Lamport, 1978; Duda *et al.*, 1987; Ashton, 1995a).

Dans le modèle d'Ellingson et Kulpinski, la fonction $C(t)$ représente la valeur retournée par une horloge C au temps réel t. Cette fonction est présumée continue et différentiable deux fois. Ses dérivées premières et secondes sont notées $C\prime(t)$ et $C\prime\prime(t)$. Une horloge parfaite est telle que $C(t) = t$. Une horloge est *correcte* au temps t_0 si $C(t_0) = t_0$. Une horloge est *fidèle* au temps t_0 si $C'(t_0) = 1$. Une horloge est *stable* au temps t_0 si $C''(t_0) = 0$. Une horloge parfaite est donc correcte, fidèle et stable. Par opposition la différence entre une horloge physique et une horloge parfaite peut être caractérisée par l'erreur suivante :

$$\Delta(t) = \alpha(t_0) + \beta(t_0)(t - t_0) + \delta(t - t_0)^2 + \epsilon(t) \tag{2.1}$$

Selon les sources, la nomenclature des termes de l'équation diverge, nous allons utiliser la suivante (et nous notons entre parenthèses les autres termes couramment utilisés) :

$\Delta(t)$ décalage au temps t (*offset, time offset*)

$\alpha(t_0)$	décalage initial
$\beta(t_0)$	décalage fréquentiel (*skew, drift, time offset rate, frequency offset*)
δ	dérive fréquentielle (*drift, drift fluctuation, frequency offset rate, frequency change rate, frequency drift*)
$\epsilon(t)$	autres facteurs, particulièrement des perturbations aléatoires

Lors d'une mesure sur un intervalle, plus la durée de celui-ci est courte, plus grand est le nombre de termes d'ordre supérieur de l'équation 2.1 qui peuvent être ignorés sans affecter la précision du résultat de manière notable. La principale contribution du modèle d'horloge est la justification de cette réduction en complexité.

Première approximation

Le taux de dérive fréquentielle pour les oscillateurs au quartz typiquement utilisé dans les ordinateurs est de l'ordre de 1 partie par 10^7 à 1 partie par 10^9 par jour (Ellingson et Kulpinski, 1973). Tous les algorithmes de synchronisation en temps absolu qui vont être présentés font l'approximation que les termes du taux de dérive fréquentielle et des autres facteurs peuvent être ignorés pour des intervalles de mesure de quelques dizaines de minutes. L'équation d'erreur devient alors :

$$\Delta(t) = \alpha(t_0) + \beta(t_0)(t - t_0) \tag{2.2}$$

Avec cette simplification, le décalage entre une horloge parfaite et une horloge physique peut donc être modélisé par une fonction linéaire. Selon les taux de dérive fréquentielle typique, ceci engendre une erreur entre .3 et 30 nanosecondes pour un intervalle de 10 minutes. Des intervalles notablement plus longs peuvent être modélisés par une succession de fonctions linéaires.

Une étude plus appronfondie de la stabilité fréquentielle a permis de séparer le taux de dérive fréquentielle en trois composantes (Mills, 1994) :

1. le bruit, d'intervalle jusqu'à une minute ; celui-ci est dû entre autres à la vibration et l'instabilité de l'alimentation électrique
2. la stabilité à court terme, d'intervalle jusqu'à une heure ; celle-ci est due entre autres aux changements de température ambiante
3. la stabilité à long terme ; celle-ci est due entre autres aux changements de masse du cristal, attribuables aux échanges gazeux avec son environnement

Le deuxième facteur joue un rôle prépondérant dans les ordinateurs, qui, pour des raisons d'économie, ne peuvent utiliser des cristaux à température contrôlée. L'étude a conclu que

le meilleur intervalle d'intégration à utiliser pour compenser les instabilités fréquentielles est d'environ 1000 secondes.

Deuxième approximation

Plusieurs algorithmes font aussi l'approximation que le décalage fréquentiel peut être ignoré pour des intervalles encore plus courts, de quelques dizaines de millisecondes. Le décalage fréquentiel maximal pour des oscillateurs au quartz étant de l'ordre de 1 partie par 10^6, ceci engendre donc une erreur maximale de 10 nanosecondes par 10 millisecondes.

Il est aussi possible d'appliquer ces deux approximations au décalage entre deux horloges physiques. On choisit une des horloges comme étant la référence et l'on fait l'hypothèse que les deux horloges se comportent selon la même approximation. C'est ce qui est fait dans les algorithmes de synchronisation qui vont suivre et qui évaluent le décalage entre les horloges de deux noeuds.

2.2.2 Algorithmes de synchronisation en ligne

À la suite des travaux de Lamport, plusieurs chercheurs ont été inspirés à publier des algorithmes se basant sur l'échange de messages pour synchroniser les horloges de machines reliées en réseau. Ces algorithmes ont généralement pour méthodologie d'évaluer le décalage entre l'horloge locale et l'horloge d'une machine de référence par l'échange de quelques messages réseau puis de faire des ajustements à l'horloge locale. Les algorithmes fonctionnent en boucle pour maintenir la synchronisation.

Algorithme de Cristian

Une percée dans ce domaine a été effectuée par Cristian en 1989 (Cristian, 1989). Cristian a proposé un algorithme probabiliste pour lire l'horloge d'une autre machine de façon aussi précise que désiré tout en tolérant des délais réseau aléatoires et illimités. La pierre angulaire de cette méthode est la stipulation de la meilleure estimation C_Q^P à faire par la machine locale P lors de la lecture par échange de messages de l'horloge de la machine de référence Q

$$C_Q^P(T, D) = T + D(1 + 2\rho) - min \cdot \rho \tag{2.3}$$

ainsi que l'erreur maximale possible e

$$e = D(1 + 2\rho) - min \tag{2.4}$$

avec T la valeur de l'horloge de référence retournée par Q, D la moitié du délai d'aller-retour mesuré par P, ρ la dérive maximale des oscillateurs et min le temps de transfert minimal du réseau.

Network Time Protocol

Certains algorithmes et outils de synchronisation utilisent les équations 2.3 et 2.4 pour calculer les corrections à appliquer à l'horloge locale. Ces équations sont particulièrement bien adaptées à un réseau local homogène et rapide. Ceci n'est toutefois pas le cas du réseau Internet, sur lequel le système de synchronisation prévalent est aujourd'hui NTP, qui désigne à la fois l'outil et le protocole. Celui-ci est aussi basé sur un échange de messages, tel qu'il est illusté à la figure 2.1.

En faisant l'hypothèse que la durée de l'échange est assez courte pour pouvoir ignorer le décalage fréquentiel ainsi que la dérive fréquentielle entre les deux horloges (deuxième approximation du modèle d'horloge), il est possible d'évaluer le décalage θ de l'horloge C_i par rapport à C_j ainsi que le délai d'aller-retour δ à l'aide des estampilles temporelles sur l'envoi et la réception des messages de l'échange (Mills, 1994) :

$$\theta = \frac{(t_2 - t_1) + (t_3 - t_4)}{2} \tag{2.5}$$

$$\delta = (t_2 - t_1) - (t_3 - t_4) \tag{2.6}$$

La valeur $\delta/2$ peut servir de marge d'erreur sur θ.

Il a été montré par la suite que ces évaluations correspondent aux estimateurs du maximum de vraisemblance lorsque les délais d'aller et de retour sont symétriques (réseau « isotropique ») et distribués selon une loi exponentielle (Jeske, 2005).

En plus de cette méthode pour évaluer le décalage, NTP comporte plusieurs algorithmes pour « discipliner » l'horloge locale, c'est-à-dire l'ajuster progressivement et la garder en synchronisation. NTP tente également d'optimiser les intervalles de resynchronisation afin de diminuer le trafic réseau.

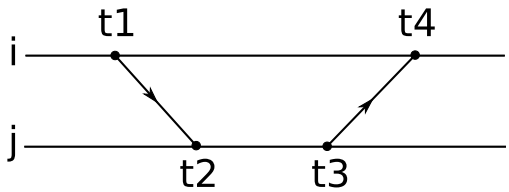


Figure 2.1 Échange de deux messages

L'amplitude du décalage entre une horloge synchronisée par NTP et sa référence varie grandement selon la stabilité de l'oscillateur local, la charge de la machine et les caractéristiques du réseau utilisé (latence et gigue). Dans une première expérience, dont les résultats sont reproduits à la figure 2.2, des mesures prises sur un réseau local révèlent un décalage oscillant entre -200 et 200μs (Mills, 2006). Ces oscillations sont quasi périodiques avec une période d'environ cinq heures. Pour cette expérience, une machine de classe Pentium 4 autrement inutilisée se synchronisait à un serveur de temps dédié situé sur le même segment d'un réseau Ethernet 100Mbs. Ceci représente donc des conditions « idéales ». Lors de ces mesures, la latence bidirectionnelle du réseau oscillait entre 700 et 860μs.

À l'opposé, dans une deuxième expérience, des mesures prises sur une variété de machines en réseau local et sur Internet révèlent les résultats présentés au tableau 2.1. Une caractéristique qui ressort de ces résultats est que les décalages moyen et maximal sont très différents l'un de l'autre. La figure 2.3 illustre le décalage entre un client NTP et sa référence sur une période de cinq jours. Bien que NTP ait une bonne stabilité à long terme (un décalage moyen faible), les données du tableau 2.1 ainsi que la figure 2.3 montrent que chaque horloge a un comportement individuel présentant i) des oscillations locales avec ii) de forts pics occasionnels (un décalage maximal de plusieurs ordres plus grand que le décalage moyen).

Ce genre de comportement (observé à différentes ampleurs dans les deux expériences mentionnées) est mal adapté à la synchronisation de traces pour deux raisons :

1. En se fiant à un ensemble de telles horloges locales, les traces, vues d'un haut niveau, présenteraient un degré de synchronisation correspondant au décalage moyen. Plusieurs auteurs s'entendent pour dire que ce degré d'exactitude est insuffisant pour des systèmes de traçage à haut débit (Doleschal *et al.*, 2008).
   Des algorithmes de synchronisation en ligne plus précis sont disponibles, notamment le Precision Time Protocol, standardisé sous la norme IEEE 1588. Ce protocole, conçu

Tableau 2.1 Charactéristiques d'homologues NTP typiques. Extrait de Mills (1994)

Serveur NTP	Décalage (ms)		
	moyen	RMS	maximal
Réseau local			
pogo-fddi.udel.edu	0.001	0.059	1.643
pogo.udel.edu	0.091	0.057	1.588
cowbird.udel.edu	-0.098	0.238	2.071
Internet			
time_a.timefreq.bldrdoc.gov	-1.511	1.686	80.567
swisstime.ethz.ch	3.102	4.533	97.291
swifty.dap.csiro.au	2.364	56.700	3944.471

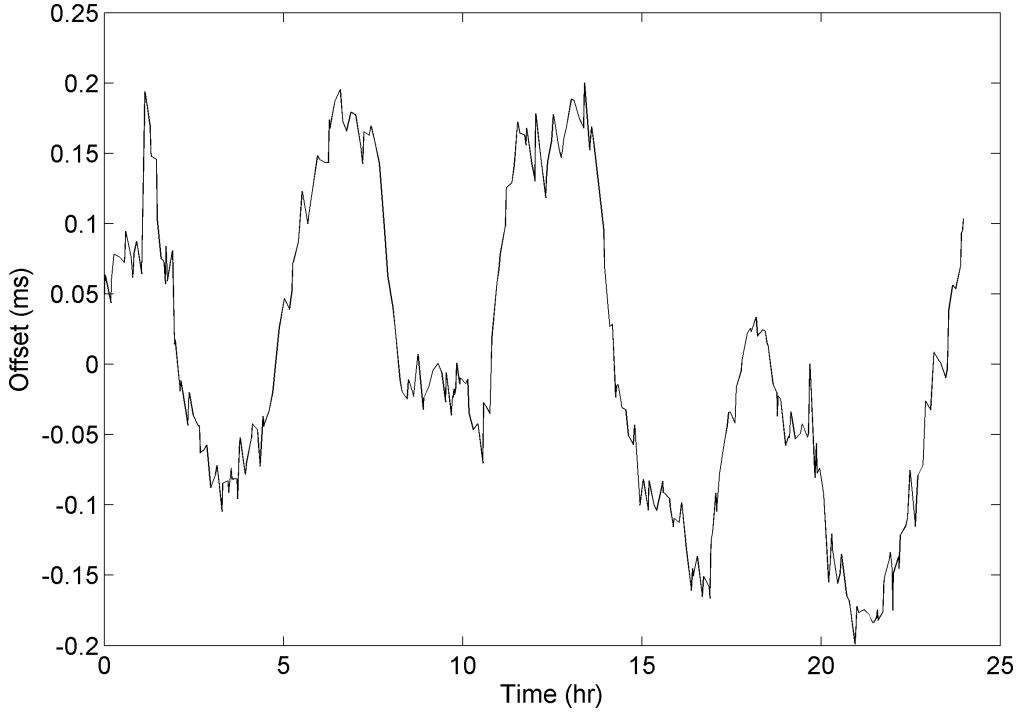


Figure 2.2 Décalage entre deux machines sur le même segment d'un réseau local. Repris de Mills (2006)

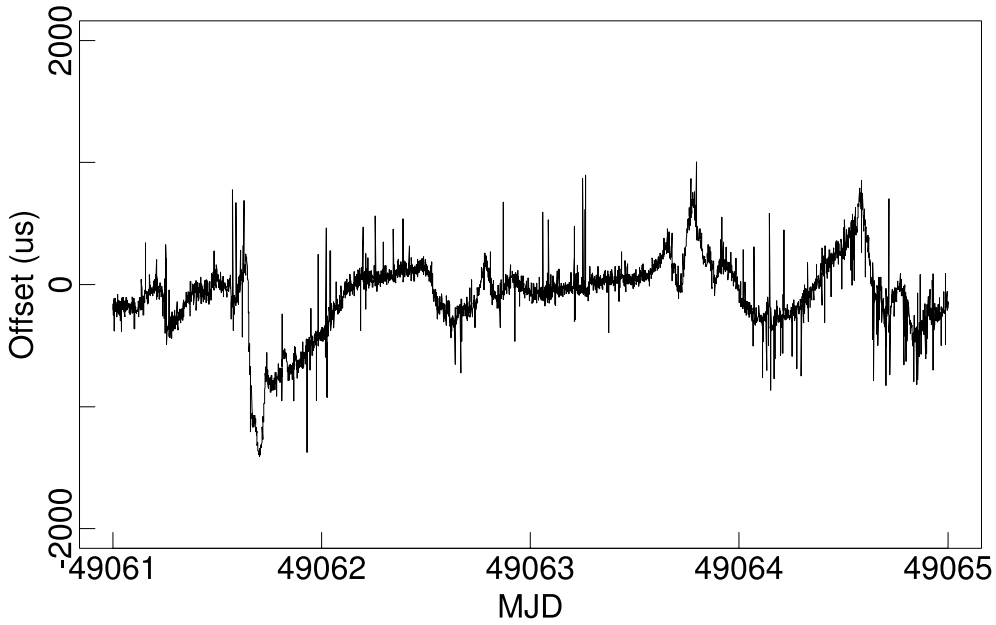


Figure 2.3 Décalage entre deux machines connectées par un réseau local multisegment. Repris de Mills (1994)

pour les réseaux d'instrumentation, est facilement capable d'un décalage moyen en deçà de 100 ns (Eidson *et al.*, 2002). Une telle exactitude nécessite toutefois du matériel spécialisé, sans quoi le décalage moyen est comparable à celui obtenu avec NTP (Correll *et al.*, 2005). Ceci en limite donc grandement l'utilité.

2. À un plus bas niveau, l'ordre partiel entre des événements de traces distinctes, relié à des relations dont la durée est inférieure à l'amplitude des oscillations locales (*i*), présenterait plusieurs inversions. Même des relations de plus longue durée présenteraient des inversions occasionnelles dues aux pics (*ii*).

Un effet secondaire de la synchronisation en ligne, qui est reconnu dans la littérature, est qu'elle introduit du trafic supplémentaire dans le système tracé. Ceci peut potentiellement modifier le comportement de ce que l'on tente de mesurer et donc nuire à la fidélité du système de traçage (Ashton, 1995a; Scheuermann *et al.*, 2009).

Un inconvénient complémentaire à ce dernier est que, dans certains cas, une horloge système contrôlée par NTP peut être non disponible, inaccessible ou trop coûteuse à lire. C'est le cas par exemple de l'architecture Cell : « A major impediment to implementing tracing on Cell is the absence of a common clock that can be accessed at low cost from all cores. The OS clock is costly to access from the auxiliary cores and the hardware timers cannot be simultaneously set on all the cores. » (Biberstein *et al.*, 2008)

Synchronisation assistée par matériel

Une approche différente pour réaliser la synchronisation en ligne consiste à utiliser du matériel spécialisé. Il est possible de distribuer un signal de temps à plusieurs noeuds à partir d'une seule horloge en les reliant par un circuit dédié. C'est le cas par exemple du super-ordinateur IBM Blue Gene/L. Celui-ci comporte une horloge maître distribuée à chaque bâti puis à chaque noeud par l'intermédiaire de câblage et de séparateurs dédiés (Coteus *et al.*, 2005). À une échelle plus réduite, certaines configurations expérimentales utilisent des câbles de même longueur reliant une horloge externe à un port d'entrée/sortie de chaque noeud (Marouani et Dagenais, 2008). Pour des noeuds éloignés, il est possible de se fier à un signal de temps externe de haute précision tel que celui émis par les satellites GPS ou la station de radio haute fréquence WWVB (Mills et Kamp, 2000). De telles configurations peuvent résulter en une synchronisation avec un décalage moyen inférieur à 1 μs et maximal inférieur à 15 μs. Malgré leur précision enviable, les méthodes de synchronisation assistées par matériel sont peu applicables. Tout comme PTP, celles-ci ont toutes l'inconvénient de nécessiter des circuits ou des récepteurs spécialisés. D'un point de vue idéologique, elles ont aussi l'inconvénient de transformer la nature du système *distribué* pour le rendre dépendant d'une horloge *centralisée* (Haddad, 1988).

Il semble donc que l'utilisation de la synchronisation en ligne dans le contexte du traçage distribué soit limitée par l'exactitude et la précision insuffisantes dans la plupart des cas, par les perturbations que cela encourrait sur le système tracé ou tout simplement par son indisponibilité. Malgré cela, une familiarité avec la synchronisation en ligne va être bénéfique pour le travail qui va suivre sur la synchronisation hors ligne. D'une part, la performance de la synchronisation en ligne peut servir de comparaison avec la synchronisation hors ligne. D'autre part, certains algorithmes et méthodes peuvent être réutilisés (Ashton, 1995a).

2.2.3 Algorithmes de synchronisation hors ligne

Alors que la synchronisation en ligne se doit généralement d'utiliser des ressources modestes, en faisant la synchronisation dans une étape de traitement ultérieure à l'enregistrement de la trace, il est possible d'utiliser des ressources de calcul à plein débit. De plus, la synchronisation hors ligne peut tenir compte, en un point, non seulement des informations passées, mais aussi futures. Ces avantages laissent miroiter la possibilité d'algorithmes de synchronisation plus exacts.

Représentation des messages

Tout comme pour le temps logique et les algorithmes en ligne, la grande majorité des algorithmes de synchronisation hors ligne est basée sur des relations d'ordre partiel. Ces relations découlent généralement de l'observation qu'un message n'est jamais reçu avant d'avoir été envoyé.

En reprenant la notation de Duda *et al.* (1987), on suppose deux machines, A et B. En utilisant l'horloge de A comme référence et en la supposant parfaite, pour simplifier les calculs, mais sans perte de généralité, on peut écrire $C_A(t) = t$. En reprenant l'équation 2.2 et en notant tout simplement le décalage initial entre A et B $\alpha(t_0)$ par α et le décalage fréquentiel $\beta(t_0)$ par β on peut écrire

$$C_B(t) = \alpha + \beta t \tag{2.7}$$

La synchronisation des deux horloges et, par extension, des traces d'événements prises individuellement sur A et B revient donc à l'estimation des paramètres α et β.

En notant les temps d'envoi ou de réception relatifs à l'horloge A pour le message i par t_i, les temps relatifs à l'horloge B par θ_i, les délais de communication par τ_i, les envois (avec l'exposant S pour *sending*) comme étant de A vers B et les réceptions (avec l'exposant R pour *receiving*) comme étant dans le sens inverse, nous avons les relations suivantes pour

chaque message :

$$\theta_i^S = \alpha + \beta\left(t_i^S + \tau_i^S\right) \quad (2.8)$$
$$\theta_i^R = \alpha + \beta\left(t_i^R - \tau_i^R\right) \quad (2.9)$$

L'envoi et la réception des messages peuvent être représentés sur des axes temporels qui forment une transposition naturelle du concept de trace, tel qu'illustré à la figure 2.4. Plusieurs algorithmes de synchronisation hors ligne peuvent toutefois être interprétés plus facilement à l'aide d'une représentation des messages comme des points sur le plan. Cette représentation est illustrée à la figure 2.5, sur laquelle les messages de la figure 2.4 ont été transposés. Avec cette représentation, un message a pour coordonnée en abscisse la valeur de l'horloge de A au moment où cette machine l'a perçu et en ordonnée la valeur de l'horloge de B.

La droite (1) représente l'équivalence que l'on aurait entre deux horloges parfaites. La droite (2) (qui est inconnue) de la figure 2.5 représente l'équivalence exacte entre les temps C_A et C_B en tout moment selon l'équation 2.7. Si les délais de communication étaient nuls, tous les points feraient partie de cette droite. Avec des délais de communication non nuls, les envois (●) se retrouvent en haut de la droite et les réceptions (■) en dessous. Les points vont donc avoir tendance à former deux « nuages » de part et d'autre de la droite. Les prochains algorithmes qui vont être présentés utilisent la disposition des points sur le plan pour estimer les paramètres de la droite (2).

Algorithme de synchronisation basé sur la régression linéaire

Le travail fondateur de la synchronisation hors ligne est un article publié en 1987 proposant la représentation sur le plan décrite à la section précédente ainsi que deux algorithmes pour estimer la droite (2) (Duda *et al.*, 1987). Le premier de ces algorithmes consiste à effectuer une régression linéaire. À l'aide de la méthode des moindres carrés, on trouve une droite passant approximativement par l'ensemble des points. Avec n messages considérés sans égard

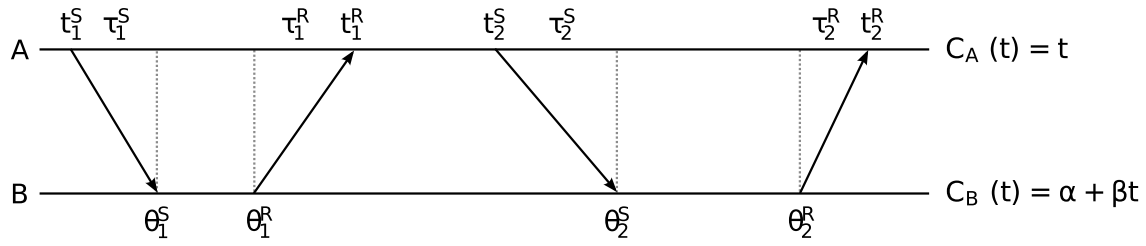


Figure 2.4 Représentation des messages sous forme de traces. Repris de Duda *et al.* (1987)

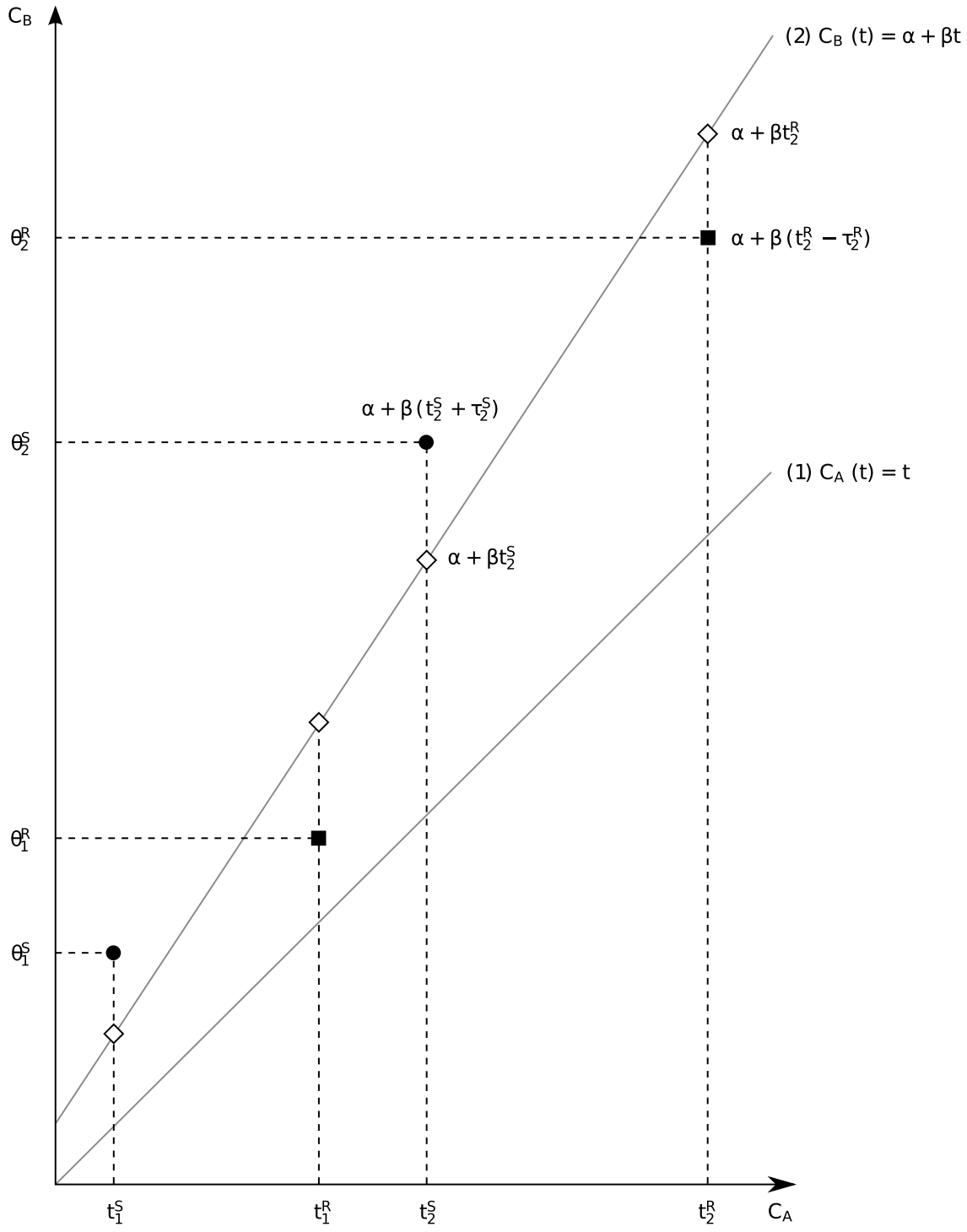


Figure 2.5 Représentation des messages sur le plan. Repris de Duda *et al.* (1987), avec ajouts

à la direction de transmission, les estimateurs pour les paramètres de la droite sont

$$\hat{\beta} = \frac{\sum_{i=1}^{n}(t_i - \bar{t})(\theta_i - \bar{\theta})}{\sum_{i=1}^{n}(t_i - \bar{t})^2} = \frac{n\sum_{i=1}^{n} t_i\theta_i - \sum_{i=1}^{n} t_i \sum_{i=1}^{n} \theta_i}{n\sum_{i=1}^{n} t_i^2 - \left(\sum_{i=1}^{n} t_i\right)^2} \tag{2.10}$$

$$\hat{\alpha} = \bar{\theta} - \bar{t}\hat{\beta} = \frac{\sum_{i=1}^{n} t_i^2 \sum_{i=1}^{n} \theta_i - \sum_{i=1}^{n} t_i \sum_{i=1}^{n} t_i\theta_i}{n\sum_{i=1}^{n} t_i^2 - \left(\sum_{i=1}^{n} t_i\right)^2} \tag{2.11}$$

la variance estimée des résidus observés

$$\hat{\sigma}_\epsilon^2 = \frac{1}{n-2}\sum_{i=1}^{N}\left(\theta_i - \hat{\theta}_i\right)^2 \tag{2.12}$$

ainsi que l'écart-type des estimateurs

$$s_{\hat{\alpha}} = \hat{\sigma}_\epsilon \sqrt{\frac{1}{n} + \frac{\bar{t}^2}{\sum_{i=1}^{N}(t_i - \bar{t})^2}} \tag{2.13}$$

$$s_{\hat{\beta}} = \frac{\hat{\sigma}_\epsilon}{\sqrt{\sum_{i=1}^{N}(t_i - \bar{t})^2}} \tag{2.14}$$

À partir des écarts-types, l'algorithme spécifie deux façons d'évaluer la précision de l'estimation. Si l'on considère que les résidus $\theta_i - \hat{\theta}_i$ sont distribués selon une loi normale, on obtient des intervalles de confiance à $100\,(1-\chi)\,\%$ pour les deux paramètres

$$\hat{\alpha} \pm t_{\frac{\chi}{2},n-2}\, s_{\hat{\alpha}} \tag{2.15}$$

$$\hat{\beta} \pm t_{\frac{\chi}{2},n-2}\, s_{\hat{\beta}} \tag{2.16}$$

où t représente la fonction de répartition de la loi de Student à $k = n-2$ degrés de liberté.

La distribution des résidus correspond à une mise à l'échelle selon $\hat{\beta}$ de la distribution des délais de transmission. Les délais de transmission réseau ont typiquement une distribution

semblable à celle présentée à la figure 2.6. Ceci ne correspond pas à une distribution normale. Il est donc proposé d'utiliser des intervalles de confiance plus grands, obtenus à partir de l'inégalité de Bienaymé-Tchebychev

$$\hat{\alpha} \pm \frac{s_{\hat{\alpha}}}{\sqrt{\chi}} \tag{2.17}$$

$$\hat{\beta} \pm \frac{s_{\hat{\beta}}}{\sqrt{\chi}} \tag{2.18}$$

L'inconvénient principal de la méthode de la régression linéaire est qu'elle ne garantit pas que l'ordre partiel des événements va être respecté. Comme les envois et les réceptions sont traités indifféremment les uns des autres, il est possible que la droite de régression passe au dessus de certains points correspondant à des envois ou vice-versa pour des réceptions. Un autre inconvénient de la méthode est qu'elle ne fournit aucune garantie de précision, aucune marge d'erreur. On ne peut qu'obtenir une estimation de celle-ci.

Régression robuste

La régression linéaire par la méthode des moindres carrés repose sur un ensemble d'hypothèses. Une de ces hypothèses est l'homoscédasticité des résidus, c'est-à-dire que la variable dépendante qui est étudiée présente une variance constante. Ceci n'est toutefois pas le cas des délais de transmission réseau (Proebstel, 2008). L'échantillon de données à traiter est hétéroscédastique : la variance du temps de transmission est influencée par certains facteurs variant d'un message à l'autre, notamment, sa taille. L'hétéroscédasticité d'un jeu de données utilisé avec la méthode des moindres carrés tend à engendrer une sous-estimation de la variance des estimateurs, qui est ensuite utilisée dans la détermination des intervalles de confiance.

Un autre facteur nuisible à l'utilisation de la méthode des moindres carrés est la présence d'observations aberrantes. Occasionnellement, certains messages vont comporter un délai

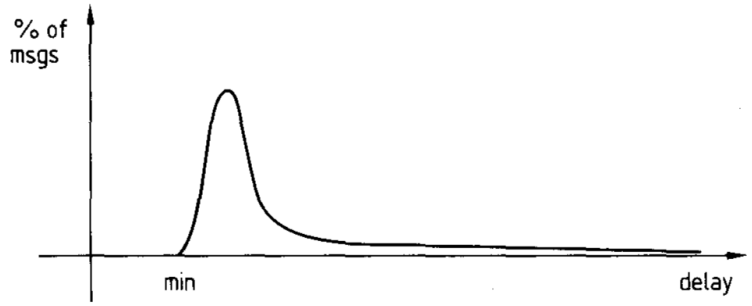


Figure 2.6 Distribution typique des délais de transmission réseau. Repris de Cristian (1989)

beaucoup plus élevé, dû par exemple à la congestion réseau ou à l'ordonnancement des tâches sur le processeur. En plus d'augmenter la variance des estimateurs, ces messages vont avoir tendance à faire dévier la droite de régression du reste des observations.

Une amélioration potentielle consiste à effectuer un filtrage préalable des messages utilisés dans la régression pour éliminer les observations aberrantes (Clément, 2006; Proebstel, 2008). Cette méthode nécessite une intervention manuelle, une adaptation selon la source présumée des observations aberrantes ou un paramétrage manuel pour établir le seuil de ce qui constitue une observation « normale ». Bien qu'il puisse éliminer les observations aberrantes, le filtrage n'a pas de fondement statistique et ne pallie pas l'hétéroscédasticité du jeu de données.

Une autre approche consiste à remplacer la régression linéaire par une variante de régression robuste. La régression robuste est une méthode de régression plus résistante aux observations aberrantes et à l'hétéroscédasticité. Lors de la synchronisation de fichiers journal distribués, Proebstel (2008) rapporte une amélioration significative lors de l'utilisation de la régression robuste par moindres carrés élagués (*least trimmed squares*). Plutôt que de minimiser la somme du carré des résidus, la régression par moindres carrés élagués minimise la somme d'un certain pourcentage (typiquement 50%) des plus petits carrés des résidus.

Deux choses sont à noter quant à la régression robuste. Premièrement, l'utilisation de la régression par moindres carrés élagués est plus exigeante en calcul que la régression par moindres carrés. Alors que cette dernière s'exécute en temps linéaire, l'application de la définition de la régression par moindres carrés élagués nécessite un temps combinatoire. Différents algorithmes approximatifs ont été développés, dont certains algorithmes itératifs qui exécutent chaque itération en temps linéaire (Rousseeuw et Driessen, 2006). Deuxièmement, bien que cette méthode puisse améliorer l'exactitude de la synchronisation, elle ne règle aucun des deux problèmes soulevés à la fin de la section 2.2.3, soit la possibilité de violation de l'ordre partiel et l'absence de garantie sur la précision.

Algorithme de synchronisation basé sur les enveloppes convexes

Tel que noté plus tôt, les points de la figure 2.5 correspondant aux messages vont avoir tendance à former deux nuages plutôt qu'une seule droite. Ceci est dû aux délais de transmission non nuls. En tentant de faire passer une droite dans le corridor entre les deux nuages, on peut s'assurer qu'aucun envoi ne se retrouve sous la droite et qu'aucune réception ne se retrouve au dessus. Il s'agit du principe du deuxième algorithme proposé dans Duda *et al.* (1987) pour estimer la droite (2) de la figure 2.5.

L'algorithme consiste à chercher les valeurs de α et β qui respectent les conditions sui-

vantes

$$\alpha + \beta t_i^S < \theta_i^S, \qquad i = 1, 2, ..., n_S \tag{2.19}$$

$$\alpha + \beta t_i^R > \theta_i^R, \qquad i = 1, 2, ..., n_R \tag{2.20}$$

Plusieurs valeurs peuvent satisfaire ces conditions, l'algorithme propose donc comme heuristique de déterminer tout d'abord les valeurs limites α_{min}, α_{max}, β_{min} et β_{max}

$$\alpha_{min}, \beta_{max} : \min_{\alpha} \max_{\beta} \quad | \quad \theta_i^R \leq \alpha + \beta t_i^R,\ i = 1, 2, ..., n_R \quad \text{et} \tag{2.21}$$

$$\alpha + \beta t_i^S \geq \theta_i^S,\ i = 1, 2, ..., n_S \tag{2.22}$$

$$\alpha_{max}, \beta_{min} : \max_{\alpha} \min_{\beta} \quad | \quad \theta_i^R \leq \alpha + \beta t_i^R,\ i = 1, 2, ..., n_R \quad \text{et} \tag{2.23}$$

$$\alpha + \beta t_i^S \geq \theta_i^S,\ i = 1, 2, ..., n_S \tag{2.24}$$

Ces valeurs définissent les droites de pentes maximale et minimale qui passent à l'intérieur du corridor entre les envois et les réceptions. Comme estimateurs, l'algorithme suggère de choisir la droite bissectrice des droites de pentes maximale et minimale

$$\hat{\beta} = \tan\left(\frac{\arctan(\beta_{min}) + \arctan(\beta_{max})}{2}\right) \tag{2.25}$$

$$= \frac{\beta_{max}\left(\sqrt{1+\beta_{min}^2} - 1\right) + \beta_{min}\left(\sqrt{1+\beta_{max}^2} - 1\right)}{\beta_{min}\beta_{max} - \left(\sqrt{1+\beta_{min}^2} - 1\right)\left(\sqrt{1+\beta_{max}^2} - 1\right)} \tag{2.26}$$

$$\hat{\alpha} = \frac{\alpha_{max}\beta_{max} - \alpha_{min}\beta_{min} - (\alpha_{max} - \alpha_{min})\,\hat{\beta}}{\beta_{max} - \beta_{min}} \tag{2.27}$$

De plus, il est certain que $\alpha \in [\alpha_{min}, \alpha_{max}]$ et $\beta \in [\beta_{min}, \beta_{max}]$.

La méthode proposée par Duda *et al.* pour déterminer les valeurs limites α_{min}, α_{max}, β_{min} et β_{max} consiste à former deux enveloppes convexes avec les messages d'envoi et de réception (Duda *et al.*, 1987). On utilise par la suite les sommets de chaque enveloppe convexe (en fait, seule une moitié supérieure ou inférieure de chaque enveloppe est nécessaire) et les inégalités 2.21 et 2.23 pour déterminer les valeurs limites. La figure 2.7 illustre un ensemble de messages, leurs enveloppes convexes, les droites de pentes maximale et minimale ainsi que la droite bissectrice finalement utilisée pour la conversion.

En termes de temps de calcul, Duda *et al.* mentionnent, à la suite d'essais sur des traces simulées, que cet algorithme est plus exigeant que l'algorithme de régression linéaire. L'algorithme ne comporte pas d'analyse de complexité, mais il est suggéré d'utiliser une marche de

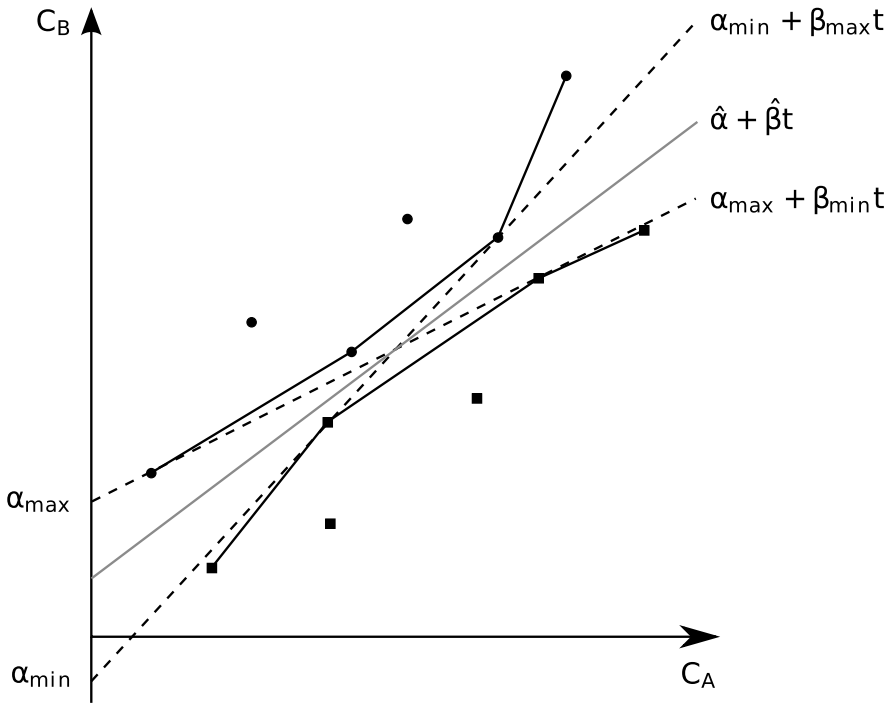


Figure 2.7 Illustration de la méthode des enveloppes convexes. Repris de Duda *et al.* (1987), avec ajouts

Jarvis pour déterminer les enveloppes convexes. Cette méthode s'exécute en temps $O(nh)$ où h représente le nombre de sommets de l'enveloppe. Ceci mène donc à une complexité $O(n^2)$ dans le pire des cas. La méthode pour déterminer les valeurs limites n'est pas détaillée.

Lors d'une implantation subséquente du même algorithme, Ashton mentionne, sans spécifier d'algorithme, que la détermination des enveloppes convexes nécessite un temps $O(n \log n)$ et que ceci borne la complexité de l'ensemble de l'algorithme (Ashton, 1995a). Encore une fois, la méthode pour déterminer les valeurs limites n'est pas détaillée, mais il est mentionné que l'implantation qui a été réalisée de l'algorithme dans son ensemble est de complexité $O(n^2)$.

Le document Haddad (1988) comporte une analyse plus approfondie de l'algorithme des enveloppes convexes. D'une part, l'auteur décrit un algorithme linéaire pour le calcul des enveloppes convexes. Il s'agit essentiellement du parcours de Graham. La complexité de cet algorithme, $O(n \log n)$, est dominée par le tri des points. Or, il est naturel d'enregistrer les traces d'exécution de telle manière que les points soient déjà classés dans chaque trace ou que l'opération de tri soit peu coûteuse. Il est donc seulement nécessaire d'effectuer la fusion des traces. Dans le cas où les points sont déjà classés, le parcours de Graham a une complexité linéaire selon le nombre de points. L'auteur décrit aussi un algorithme linéaire pour déterminer β_{min} et β_{max}. Il semble donc possible d'implanter l'ensemble de l'algorithme

en temps linéaire par rapport au nombre de messages.

Le même document fait remarquer que l'évaluation numérique de $\hat{\alpha}$ et $\hat{\beta}$ selon les équations 2.26 et 2.27 comporte des inconvénients :

- pour déterminer $\hat{\beta}$ il est nécessaire d'utiliser des fonctions trigonométriques ou d'évaluer quatre racines carrées
- pour déterminer $\hat{\alpha}$, la différence $\beta_{max} - \beta_{min}$ au dénominateur tend vers 0

Des équations alternatives sont proposées

$$\hat{\beta} = \frac{\beta_{min}\beta_{max} - 1 + \sqrt{1 + \beta_{min}^2\beta_{max}^2 + \beta_{max}^2 + \beta_{min}^2}}{\beta_{min} + \beta_{max}} \tag{2.28}$$

$$\hat{\alpha} = \alpha_{min} + \frac{\alpha_{max} - \alpha_{min}}{2} \cdot \frac{\hat{\beta}^2 + 1}{\hat{\beta}\beta_{min} + 1} \tag{2.29}$$

$$= \alpha_{max} - \frac{\alpha_{max} - \alpha_{min}}{2} \cdot \frac{\hat{\beta}^2 + 1}{\hat{\beta}\beta_{max} + 1} \tag{2.30}$$

Finalement, ce document comporte une analyse de convergence des estimateurs qui révèle que lorsque n_R et n_S tendent vers l'infini, $\hat{\beta}$ converge vers β et $\hat{\alpha}$ converge vers :

$$\alpha + \beta\frac{\tau_{min}^S - \tau_{min}^R}{2}$$

Lors de l'application de l'algorithme basé sur les enveloppes convexes, il peut arriver qu'il soit impossible de faire passer une droite dans le corridor entre les enveloppes. Dans ce cas, l'algorithme ne peut pas fournir de solution. Ceci peut être attribué à la violation d'au moins une des hypothèses de l'algorithme, notamment l'approximation que les termes du taux de dérive et des autres facteurs peuvent être ignorés dans le modèle d'horloge, tel que décrit à la section 2.2.1. Ceci peut aussi être attribué à une granularité trop grossière des estampilles temporelles par rapport aux délais de transmission.

L'algorithme basé sur les enveloppes convexes permet donc de palier aux lacunes de l'algorithme basé sur la régression linéaire. Il garantit le respect de l'ordre partiel entre les événements et offre une garantie sur la précision. Il semble toutefois que cet algorithme soit plus exigeant en temps de calcul que la régression linéaire.

Utilisation du délai de transmission minimal

Tel qu'illustré à la figure 2.6, la distribution des délais de transmission réseau comporte un certain délai minimal, $\tau_{min} > 0$. Ce délai peut être intégré dans les algorithmes de régression et d'enveloppes convexes pour en augmenter la précision (Ashton, 1995a). On additionne $1/2\ \tau_{min}^S$ aux estampilles temporelles de A et on soustrait $1/2\ \tau_{min}^S$ des estampilles temporelles

de B lors des envois. On fait l'inverse lors des réceptions.

D'un point de vue géométrique, cette correction réduit l'espace entre les deux nuages de points sur le plan. Il s'agit d'une transformation bénéfique à l'algorithme de régression puisque les points corrigés vont être distribués autour d'une seule droite plutôt que deux droites parallèles. La transformation est aussi bénéfique pour l'algorithme basé sur les enveloppes convexes. En réduisant l'espace entre les points d'envoi et de réception, on réduit la largeur du corridor entre les enveloppes convexes. Ceci a pour effet de réduire l'angle entre les droites de pentes maximale et minimale et donc de réduire la marge d'erreur.

D'un point de vue mathématique, ceci transforme la distribution de τ_i, les délais de transmission. La distribution a alors une fonction de masse f_τ telle que $f_\tau(x) > 0, \;\; x > 0$. Ceci garanti à son tour que les différences $\alpha_{max} - \alpha_{min}$ et $\beta_{max} - \beta_{min}$ dans l'algorithme des enveloppes convexes convergent vers 0 lorsque le nombre de messages n_S et n_R tend vers l'infini (Duda *et al.*, 1987). À l'opposé, sans cette correction, la distribution est telle que $f_\tau(x) = 0, \;\; 0 \leq x \leq \tau_{min}$ et $\beta_{max} - \beta_{min} \to 0$ mais $\alpha_{max} - \hat{\alpha} \to \tau_{min}$ et $\hat{\alpha} - \alpha_{min} \to \tau_{min}$.

La correction des points en fonction du délai de transmission minimal exacerbe le risque que l'algorithme ne puisse pas fournir de solution. En proposant cette correction, Ashton a aussi proposé une modification de la deuxième partie de l'algorithme, la détermination des valeurs limites, afin de fournir une solution même lorsqu'aucune droite ne peut traverser le corridor.

Cette modification consiste à calculer une erreur totale pour chacune des droites de l'ensemble des droites passant par un point dans chaque enveloppe convexe. L'erreur totale est calculée comme étant la somme des distances verticales entre la droite et chaque point qui fait partie d'une enveloppe convexe et qui est du mauvais côté de la droite. Ce type d'erreur est semblable à celui utilisé dans la régression par moindres carrés, toutefois on additionne la valeur absolue des résidus plutôt que leur carré. Si aucune droite n'a une erreur nulle, l'algorithme retient la droite avec la plus faible erreur. L'implantation de l'algorithme n'est pas détaillée, mais une analyse de complexité sommaire indique qu'un tel algorithme doit être au moins d'ordre $O(h^2)$ puisqu'il faut tester une droite pour chaque paire de points faisant partie des enveloppes convexes.

Un autre groupe d'auteurs a également proposé d'ajuster les estampilles temporelles lors de l'utilisation de l'algorithme des enveloppes convexes (Jezequel et Jard, 1996). Ils suggèrent d'augmenter l'espace entre les deux enveloppes convexes lorsque celles-ci se chevauchent et de réduire l'espace lorsque possible afin d'augmenter artificiellement la précision.

La correction des estampilles temporelles permet d'augmenter la précision des algorithmes de régression et d'enveloppes convexes. Pour ce faire, il est toutefois nécessaire de connaître le délai minimal de transmission, τ_{min}. Bien qu'il soit possible d'utiliser un programme de

test de performance réseau pour évaluer une approximation de celui-ci, il serait beaucoup plus pratique de pouvoir l'évaluer à partir des estampilles temporelles des messages tracés. La section suivante présente une méthode permettant de le faire.

Utilisation d'échanges de messages

Les algorithmes de synchronisation hors ligne présentés jusqu'à maintenant utilisent tous les messages individuellement. Un message est constitué de deux événements qui sont appariés : un événement d'envoi d'un paquet réseau dans une trace et un de réception du même paquet dans une autre trace. À l'opposé, les deux algorithmes de synchronisation en ligne présentés à la section 2.2.2 utilisent des échanges de messages, tel que celui illustré à la figure 2.1. Or, il est possible de transposer ce concept à la synchronisation hors ligne. Il faut alors non seulement former des messages, mais également apparier des messages pour former des échanges de messages.

En formant des échanges avec les événements contenus dans les traces, il est possible, à l'aide de l'équation 2.6, d'évaluer les délais d'aller-retour. En faisant ceci systématiquement, on peut trouver l'échange ayant le délai le plus petit et utiliser la moitié de cette valeur comme approximation de τ_{min}. Ceci peut donc servir pour faire les corrections décrites à la section précédente.

L'utilisation d'échanges de messages, prévalente dans les algorithmes de synchronisation en ligne, permet d'évaluer le délai de transmission. La section suivante montre qu'il est possible d'emprunter des concepts supplémentaires à la synchronisation en ligne et de les appliquer à la synchronisation hors ligne.

Algorithme de synchronisation à l'aide de deux points

L'algorithme de Cristian pour la synchronisation en ligne, décrit à la section 2.2.2, consiste à générer des échanges de messages jusqu'à en obtenir un ayant un délai d'aller-retour suffisamment faible. Dans ses travaux, Ashton s'est inspiré de cet algorithme pour créer un nouvel algorithme de synchronisation hors ligne. Comme il n'est pas possible de générer des échanges de messages a posteriori, l'algorithme consiste plutôt à analyser l'ensemble des échanges de messages dans deux traces pour choisir deux échanges qui ont un faible délai d'aller-retour. À partir de ces deux échanges, deux valeurs de décalage sont calculées. Celles-ci correspondent essentiellement à deux points sur le plan de la figure 2.5. On détermine le décalage initial et la dérive entre deux horloges à l'aide de la droite passant par ces deux points. La plus grande partie de l'algorithme concerne la façon dont on doit choisir les deux échanges à utiliser.

Après avoir identifié tous les échanges de messages entre les deux traces, on utilise l'équa-

tion 2.6, provenant également de la synchronisation en ligne, pour calculer le délai d'aller-retour de chacun. Deux de ces échanges serviront en définitive à calculer deux décalages à l'aide d'un développement de l'équation 2.5. Or, à défaut d'avoir plus d'information, la moitié du délai d'aller-retour représente une borne sur l'erreur commise en calculant un décalage de cette façon. On cherche donc à choisir deux échanges qui ont un délai d'aller-retour faible.

La courte durée des deux échanges n'est cependant pas le seul critère. En conservant la notation décrite à la section 2.2.3 et en utilisant les indices $i1$ et $i2$ pour les deux échanges choisis et l'indice $i3$ pour un échange quelconque, possiblement $i1$ ou $i2$, les estimateurs sont déterminés de la façon suivante

$$\hat{\beta} = \frac{\left(\theta^S_{i1} + \theta^R_{i1}\right) - \left(\theta^S_{i2} + \theta^R_{i2}\right)}{\left(t^S_{i1} + t^R_{i1}\right) - \left(t^S_{i2} + t^R_{i2}\right)} \tag{2.31}$$

$$\hat{\alpha} = \left(\theta^S_{i3} + \theta^R_{i3}\right) - \hat{\beta}\left(\theta^S_{i3} + \theta^R_{i3}\right) \tag{2.32}$$

La direction de chaque échange choisi peut être différente sans influencer l'algorithme. Une marge d'erreur, $\hat{\beta} \pm \Delta\hat{\beta}$, est obtenue de la façon suivante

$$\Delta\hat{\beta} = \frac{\delta_{i1} + \delta_{i2}}{2\left|\left(t^S_{i1} + t^R_{i1}\right) - \left(t^S_{i2} + t^R_{i2}\right)\right|} \tag{2.33}$$

où δ est obtenu à l'aide de l'équation 2.6. On constate à partir du dénominateur que le temps écoulé entre les deux échanges devrait être élevé pour améliorer la précision. Le principe général de l'algorithme consiste donc à choisir deux échanges de courte durée, mais les plus espacés possible.

La recherche exhaustive parmi toutes les paires d'échanges de messages pour trouver celle qui mène au $\Delta\hat{\beta}$ le plus petit comporte une complexité d'ordre $O\left(n^2\right)$ dans le pire des cas. L'algorithme propose donc une heuristique présentée dans l'algorithme 2.1. Cet algorithme mène à une complexité d'ordre $O\left(n\right)$ puisqu'il faut parcourir les deux traces dans leur ensemble pour identifier et sélectionner les échanges.

Tout comme pour les algorithmes de régression et d'enveloppes convexes, la précision de cet algorithme peut être améliorée si l'on tient compte du délai de transmission minimal, τ_{min}. Pour que l'équation 2.5 soit juste, il faut non seulement que le délai soit court (afin de pouvoir faire la deuxième approximation du modèle d'horloge) mais aussi que les délais d'aller et de retour soient symétriques. Si un échange présente un délai d'aller-retour $\delta \approx 2\tau_{min}$ alors les délais des deux messages doivent être presque symétriques puisqu'ils sont tous deux près du délai de transmission minimal (Ashton, 1995a). L'algorithme propose donc de raffiner

1. Diviser la période couverte par les échanges en dix intervalles de longueur égale.
2. Dans chaque intervalle, choisir les dix échanges de messages ayant la plus faible marge d'erreur sur le décalage calculé.
3. Faire une recherche exhaustive parmi ces 100 échanges de messages pour trouver celui qui mène au $\Delta\hat{\beta}$ le plus faible.
4. Déterminer $\hat{\beta}$ à partir de cet échange.
5. Déterminer $\hat{\alpha}$ à l'aide de l'échange qui a la plus faible marge d'erreur sur le décalage calculé.

ALGORITHME 2.1 Heuristique de l'algorithme de synchronisation à l'aide de deux points

l'équation 2.6 en tenant compte du délai de transmission minimal

$$\delta^{\star} = \delta - 2\tau_{min} \tag{2.34}$$

Ceci permet de déterminer l'erreur maximale commise lors du calcul d'un décalage. On utilise ce raffinement aux étapes 2 et 5 de l'algorithme 2.1. Encore une fois, τ_{min} peut être évalué à l'aide d'un programme de test de performance réseau si possible ou à l'aide de l'équation 2.6.

Bien qu'au final il n'utilise que quatre messages, cet algorithme considère tous les échanges pour choisir ces messages de manière à améliorer la précision. Inspiré par les méthodes de synchronisation en ligne, où les ressources sont limitées, cet algorithme représente peut-être un meilleur compromis entre la vitesse d'exécution et la précision que les algorithmes basés sur la régression et les enveloppes convexes.

Variations et autres algorithmes

Algorithme de synchronisation de Hofmann et Hilgers Une variation de l'algorithme de régression linéaire est décrite dans Hofmann et Hilgers (1998). Cet algorithme vise l'efficacité avant tout et divise un ensemble de traces de même durée en intervalles de longueur suffisamment courte pour pouvoir appliquer uniquement une correction du décalage sans causer de violation de l'ordre partiel. Le décalage fréquentiel est estimé par régression linéaire du décalage de chaque intervalle. Tout comme l'algorithme de régression linéaire, cet algorithme s'exécute en temps linéaire en fonction du nombre d'événements des traces. Aucune évaluation empirique n'est faite de la précision de cet algorithme.

Algorithme de synchronisation de Sirdey et Maurice Une avenue plus récente pour la synchronisation d'horloge est l'utilisation d'approches par programmation linéaire. C'est le

cas d'un raffinement de l'algorithme des enveloppes convexes qui cherche un corridor le plus large possible plutôt qu'une droite (Sirdey et Maurice, 2008). Cet algorithme, conçu pour les systèmes d'infrastructure de téléphonie GSM, cherche en priorité à évaluer le décalage fréquentiel entre deux horloges, mais il permet aussi de déterminer le décalage.

L'interprétation du problème est la même que pour l'algorithme des enveloppes convexes, seule la méthode de résolution est différente : programmation linéaire plutôt qu'approche géométrique. Le programme linéaire comportant trois variables peut être résolu en temps linéaire par rapport au nombre de contraintes, les messages. L'implantation réalisée utilise un solveur générique et offre une précision impressionnante en simulation. Une dérive de $\beta = 1 + 2 \times 10^{-8}$ est évaluée à l'aide de 120 000 messages simulés dont les délais suivent une distribution de Weibull paramétrée à partir d'observation sur un réseau local. La dérive est estimée avec une erreur moyenne de 4.51×10^{-12}. Il est toutefois à noter que les horloges simulées respectent exactement la deuxième approximation, elles n'ont donc ni taux de dérive, ni erreurs aléatoires.

Algorithme de synchronisation de Scheuermann et al. Un autre algorithme récent utilise la programmation linéaire, mais se base uniquement sur les événements de réception des messages de diffusion générale (*broadcast*) (Scheuermann *et al.*, 2009). Ceci a deux avantages. D'une part, la partie des délais de transmission commune à tous les noeuds est éliminée pour ne laisser qu'un délai d'horodatage (*timestamping*) variable d'un noeud à l'autre. D'autre part, cette approche permet la synchronisation simultanée d'un groupe de traces plutôt que d'être restreinte à des échanges entre deux machines. Naturellement, ceci a le désavantage de ne fonctionner que dans le cadre de réseaux locaux à diffusion générale.

Pour la résolution, le problème est exprimé comme un problème d'optimisation. L'algorithme cherche les paramètres de décalage et de décalage fréquentiel qui maximisent la vraisemblance des délais estimés à partir des estampilles temporelles observées. L'algorithme pose l'hypothèse que les délais d'horodatage suivent une distribution exponentielle et cherche une estimation du maximum de vraisemblance. Après transformations, il s'avère que la solution optimale minimise la somme des délais observés et est indépendante de la distribution des délais. Le problème est ainsi reformulé en tant que programme linéaire comportant un nombre de variables et de contraintes qui sont toutes deux fonction du nombre de noeuds et d'événements. Ceci rend l'application d'un solveur générique peu pratique et les auteurs décrivent leur implantation d'un solveur spécialisé basé sur la méthode du point intérieur. Celui-ci permet de résoudre en une trentaine de secondes des problèmes impliquant une centaine de noeuds ayant observé environ 100 000 événements en commun. L'algorithme ne fournit aucune garantie de fidélité ou de justesse.

Algorithme de synchronisation de Becker et al. (horloge logique contrôlée) Finalement, un dernier algorithme, l'horloge logique contrôlée, se base uniquement sur les violations d'ordre partiel pour faire des ajustements locaux aux horloges (Becker *et al.*, 2009). Ceci implique que l'algorithme ne fera pas d'ajustements aux horloges s'il n'y a pas de violations de l'ordre partiel. Cet algorithme nécessite aussi que les traces soient « faiblement » synchronisées au préalable par une autre méthode. Contrairement aux autres algorithmes hors ligne, cet algorithme ne calcule pas le décalage et le décalage fréquentiel entre les horloges. Ceci lui confère l'avantage de pouvoir fonctionner sur des durées trop longues pour pouvoir appliquer directement la deuxième approximation du modèle d'horloge. Par contre, ceci représente aussi un désavantage théorique puisque les corrections appliquées par l'algorithme ne sont pas justifiées par le modèle d'horloge. Cet algorithme peut fonctionner sur un groupe de traces plutôt que seulement un couple. Sa complexité algorithmique est dans l'ordre $O(n \log n)$.

Synchronisation d'un groupe d'horloges

La majorité des algorithmes hors ligne présentés jusqu'à maintenant (toutes les variantes d'algorithmes à l'aide de régression, d'enveloppes convexes et de deux points) permettent uniquement de synchroniser un couple d'horloges. Plusieurs systèmes distribués sont toutefois constitués de plus de deux noeuds. Une solution à cette situation fut proposée en même temps que les premiers algorithmes : synchroniser l'ensemble des horloges deux à deux et propager les facteurs d'ajustement d'horloge (Duda *et al.*, 1987).

Algorithme de propagation de Duda et al. Selon la topologie des communications, deux cas extrêmes peuvent être considérés. Le premier cas est celui où chaque noeud ne communique qu'avec ses voisins, tel que dans un réseau en anneau. Dans ce cas, les facteurs de correction d'horloge peuvent être propagés du premier noeud au deuxième, du deuxième au troisième, et ainsi de suite itérativement. Si on utilise un algorithme qui garantit le respect de l'ordre partiel, celui-ci sera respecté entre toutes les traces. Le désavantage de cette approche est que plus on s'éloigne du noeud initial dans la chaîne, plus les marges d'erreur s'agrandissent.

Le deuxième cas extrême est celui où tous les noeuds communiquent ensemble. Dans ce cas, on peut choisir un noeud de référence et calculer les facteurs de correction entre chaque autre noeud et cette référence. Par contre, même si on utilise l'algorithme des enveloppes convexes qui garantit le respect de l'ordre partiel entre le noeud de référence r et chaque autre noeud, disons i et j, il n'est pas garanti que l'ordre partiel soit respecté entre les noeuds i et j. Pour avoir cette garantie, il est nécessaire d'ajuster toutes les valeurs limites

α_{min}, α_{max}, β_{min} et β_{max}. Un algorithme pour ce faire est donné dans Duda *et al.* (1987).

Algorithme de propagation de Jézéquel Un algorithme alternatif pouvant gérer des topologies quelconques est proposé dans Jezequel (1989); Jezequel et Jard (1996). Cet algorithme utilise un arbre couvrant des plus courts chemins pour choisir la façon dont les facteurs de correction sont propagés. L'arbre est construit à partir d'un graphe dans lequel chaque noeud représente une machine du réseau. Une arête de longueur unitaire relie deux noeuds voisins, c'est-à-dire des machines qui ont échangé des messages. Si une des machines est connectée à une horloge externe, son noeud peut être utilisé comme référence initiale (il devient la racine de l'arbre), sinon, ce choix n'est pas spécifié. Selon la topologie des communications, cet algorithme va s'exécuter récursivement sur des sous-parties de l'arbre.

La complexité d'exécution de l'algorithme est bornée par la recherche des plus courts chemins entre toutes les paires de noeuds pour identifier un noeud racine et son arbre couvrant de poids minimal. En utilisant l'algorithme de Johnson, cette recherche nécessite un temps dans l'ordre de $O\left(V^2 \log V + VE\right)$, où V est le nombre de noeuds et E le nombre d'arrêtes du graphe (soit le nombre de paires de noeuds qui ont échangés des messages). En considérant uniquement le nombre de noeuds, ceci donne un ordre d'exécution de $O\left(V^3\right)$ en pire cas, pour un graphe complètement connecté. Si l'on considère plutôt le nombre d'arêtes, la complexité d'exécution de l'algorithme est linéaire. [1] De plus, si la topologie du réseau est connue d'avance, l'arbre peut être calculé statiquement. Finalement, le reste de l'algorithme de Jézéquel est dans l'ordre de $O\left(V\right)$ dans le pire des cas.

Algorithme de propagation de Hofmann et Hilgers Lors de la description de leur algorithme de régression linéaire simplifiée, Hofmann et Hilgers ont aussi proposé une façon de propager les facteurs de correction (Hofmann et Hilgers, 1998). Leur méthode peut être utilisée uniquement lorsque l'on ne considère que les facteurs de correction du décalage. La nouveauté de cet algorithme par rapport aux deux précédents est qu'il propose une façon de choisir l'horloge de référence de façon à minimiser les erreurs. L'algorithme ajuste d'abord les valeurs de α_{min} et α_{max} entre chaque noeud en parcourant l'ensemble du graphe avec un algorithme de recherche des plus courts chemins. Une fois ceci complété, la marge d'erreur des facteurs de correction entre chaque noeud est évaluée. Le noeud référence est finalement choisi comme étant celui qui peut rejoindre tous les autres noeuds avec la plus petite somme cumulative des marges d'erreur.

1. Comme $E \in O\left(M_{nb}\right)$, où M_{nb} est le nombre de messages échangés, et $M_{nb} \in O\left(T_{traces}\right)$, où T_{traces} est la taille du groupe de traces, la complexité d'éxécution de l'algorithme est linéaire en fonction de la taille du groupe de traces.

Algorithme de propagation de Clément Dans ses travaux qui utilisent l'algorithme de régression linéaire, Clément propose une autre façon de déterminer la référence et de propager les facteurs de correction (Clément, 2006). Cet algorithme construit un graphe similaire à l'algorithme de Jézéquel, mais dans lequel les arêtes ont pour poids l'écart-type estimé des résidus obtenus lors de la régression. L'algorithme effectue par la suite une recherche des plus courts chemins et détermine la référence comme dans l'algorithme de Hofmann et Hilgers. Les facteurs de correction sont propagés le long des arêtes formant les plus courts chemins.

2.2.4 Outils pour la synchronisation hors ligne

Généralement, les outils de synchronisation hors ligne sont liés au système utilisé pour tracer le système distribué, qui est à son tour lié à la bibliothèque utilisée pour le programmer. De plus, les algorithmes de synchronisation sont intégrés à des outils d'analyse et de visualisation de traces. Voici quelques outils qui permettent la synchronisation hors ligne ainsi que les algorithmes qu'ils utilisent.

Vampir

Vampir est un outil de visualisation spécialisé pour la bibliothèque de programmation distribuée MPI. Disponible depuis 1996, il s'agit d'un produit commercial développé par différents centres de recherche et universités allemands. Il comporte aussi un logiciel de traçage associé, disponible sous licence libre, VampirTrace, et peut lire les traces enregistrées par certains autres programmes de traçage. Une partie de Vampir a été acquise par Intel et sert de fondation au Intel Trace Analyzer.

La synchronisation effectuée par Vampir est très élémentaire. VampirTrace réalise deux rondes de synchronisation, une avant l'exécution du programme distribué et une autre après (Knupfer *et al.*, 2008). Dans une étape de posttraitement, l'ensemble des estampilles temporelles est ajusté linéairement. Cette approche courante est désignée comme « échantillonnage avant et après » (SBA).

Chaque ronde est composée de dix échanges entre une machine référence et chaque autre machine. La référence mesure le délai d'aller-retour de chaque échange. Celui avec le plus petit délai est utilisé avec une version simplifiée de l'équation 2.3 de l'algorithme de Cristian pour que chaque machine évalue son décalage par rapport à la référence (Technische Universität Dresden - Zentrum für Informationsdienste und Hochleistungsrechnen, 2009).

Cette méthode a l'avantage d'être simple et de nécessiter peu de ressources. Elle a par contre deux désavantages majeurs. D'une part, la précision obtenue dépend de l'exactitude des deux seules mesures effectuées. D'autre part, cette méthode ne tient pas compte de la

dérive des horloges et devient donc rapidement inadéquate pour des intervalles de traçage de plus d'une dizaine de minutes. Face aux inversions apparentes de message auxquelles leurs usagers étaient confrontés (GWT-TUD GmbH, 2008), les développeurs de Vampir fournissent maintenant une méthode de synchronisation additionnelle.

Depuis novembre 2008, la version 5.6 de VampirTrace inclut un mode de synchronisation améliorée. Plutôt que de réaliser les rondes de synchronisation uniquement au début et à la fin du programme, des rondes additionnelles sont effectuées durant l'exécution du programme (Doleschal *et al.*, 2008). Les estampilles temporelles sont ensuite ajustées linéairement dans chacun des sous-intervalles. Ceci réduit effectivement l'intervalle d'interpolation à une durée assez courte pour pouvoir faire la première approximation du modèle d'horloge.

Le désavantage d'une telle approche hybride en ligne et hors ligne est que les rondes de synchronisation intermédiaires peuvent perturber l'exécution normale du programme. Ceci est d'ailleurs explicitement mentionné dans le manuel d'utilisation de VampirTrace (Center for Information Services and High Performance Computing (ZIH), 2008). Comme les rondes de synchronisation représentent effectivement une barrière globale, celles-ci sont insérées de préférence autour de fonctions collectives MPI impliquant une barrière déjà présente dans le programme. Lors d'une expérience d'une durée de 120 secondes sur un système de 23 noeuds, dix rondes de synchronisation ont introduit un surcoût d'approximativement 6.5 s (Doleschal *et al.*, 2008).

Scalasca

Scalasca est une suite d'outils pour l'analyse de performance conçue expressément pour les systèmes distribués de grande échelle (Wolf *et al.*, 2008). Celle-ci est principalement développée au Jülich Supercomputing Centre en Allemagne. Il s'agit du successeur de KOJAK, la bibliothèque de traçage qui était aussi à l'origine de VampirTrace. La fonctionnalité principale de Scalasca est d'effectuer une analyse des états d'attente lors de l'exécution de programmes MPI et OpenMP. Les états d'attente sont composés de périodes de temps où certains processeurs ne font aucun travail utile car ils attendent d'autres processeurs. Il s'agit d'un facteur important qui limite l'utilisation maximale des ressources d'un système parallèle ou distribué.

Contrairement à Vampir qui affiche les traces sur une frise chronologique, Scalasca produit plutôt un rapport des « propriétés de performance » (métriques et patrons) du programme. Ce rapport est composé à partir de l'analyse des états d'attente qui est à son tour faite à partir de traces d'exécution enregistrées individuellement par chaque noeud du système distribué. Pour mesurer les états d'attente, il est essentiel que les traces soient synchronisées. À cette fin, Scalasca emploie l'algorithme de l'horloge logique contrôlée. Tel qu'il a été mentionné à la section 2.2.3, cet algorithme nécessite toutefois une certaine présynchronisation. Celle-ci

est réalisée à l'aide d'une approche « échantillonnage avant et après » (Becker *et al.*, 2009), tout comme le fait la version de base de VampirTrace.

La synchronisation ainsi que l'analyse sont implantées de manière distribuée et s'exécutent sur autant de noeuds que ce qui a été utilisé pour enregistrer la trace. Les traces sont chargées en mémoire principale avant le début de la synchronisation. Lors d'une série d'expériences avec un nombre de noeuds variant de 64 à 1024, ceux-ci ont synchronisé des traces contenant des événements pour un total de 2.8 à 90.5 millions de messages en 6 à 17 secondes (Becker *et al.*, 2009). En vertu de leurs mesures, les auteurs avancent que la synchronisation et l'analyse sont tributaires des entrées-sorties.

Linux Trace Toolkit Next Generation

LTTng est un traceur noyau développé à l'École Polytechnique de Montréal depuis 2005. Celui-ci a pour objectif d'aider à l'identification de problèmes de fonctionnalité et de performance liés au noyau Linux. LTTng comporte un visualiseur qui permet de consulter un ensemble de traces provenant de plusieurs machines. Il n'inclut toutefois aucune fonctionnalité pour synchroniser ces traces.

Lors de ses travaux, Clément a utilisé l'algorithme de régression linéaire avec filtrage préalable des messages pour réaliser la correction des estampilles temporelles dans un ensemble de traces LTTng (Clément, 2006). Les facteurs de correction sont propagés à l'aide de l'algorithme décrit à la section 2.2.3.

Un aspect particulier de ce travail est qu'il propose une approche pratique à l'appariement des événements de différentes traces pour identifier les messages et à l'appariement des messages pour former des échanges. Le système développé tire avantage de l'instrumentation noyau de LTTng pour analyser les paquets réseau transitant entre les machines. L'information contenue dans l'en-tête des paquets TCP est suffisante pour réaliser l'appariement. Ceci veut donc dire que le système peut synchroniser un ensemble de traces d'exécution provenant d'un système distribué sans avoir à réaliser de ronde de synchronisation et sans modifier les applications.

Plusieurs expériences portant sur la synchronisation de deux noeuds reliés par un réseau local Ethernet 100Mbps ont été réalisées. Lors d'une série d'expériences d'une durée de dix minutes avec un débit de message variable, l'écart-type estimé des résidus observés lors de la régression a varié entre 8.46×10^{-7} et 3.29×10^{-6} secondes.

Tuning and Analysis Utilities et Kernel Tuning and Analysis Utilities

Tout comme Vampir et Scalasca, TAU est une suite d'outils d'analyse de performance conçue pour les programmes parallèles et distribués. À la différence de ces deux premiers, l'intérêt principal de TAU est le profilage plutôt que le traçage. C'est-à-dire que son outil de visualisation, Paraprof, présente des résumés statistiques de l'exécution d'un programme plutôt qu'une frise chronologique des événements. Malgré tout, TAU permet aussi de générer des traces qui peuvent être exportées et consultées dans les visualiseur d'autres projets, tels Vampir ou Jumpshot (présenté à la section 2.2.4).

KTAU est le volet noyau du projet TAU (Nataraj *et al.*, 2008). Tout comme LTTng, il s'agit d'un traceur pour le noyau Linux. À la différence de ce dernier, KTAU avait toutefois pour objectif d'être utilisé sur des systèmes distribués dès sa conception. Malgré tout, KTAU ne permet pas de faire la synchronisation de traces, ceci pour deux raisons. D'une part, KTAU n'inclut pas son propre visualiseur mais utilise plutôt Paraprof. Le genre d'analyse montrée par KTAU est donc le même que pour TAU. Il s'agit plutôt de profilage qui nécessite une agrégation des traces prises sur un système distribué, mais pas une synchronisation. D'autre part, tout comme pour TAU, il est possible d'exporter les traces pour les consulter dans les visualiseurs de d'autres projets. Toutefois, dans le cas de KTAU, il n'est possible d'exporter les traces que d'un seul noeud à la fois.

TAU utilise le même code que VampirTrace pour calculer le décalage entre les horloges (il s'avère que ce code provient de KOJAK) (Department of Computer and Information Science - University of Oregon, 2009). Le système de traçage prend une mesure du décalage au début de l'exécution du programme et l'utilise pour corriger les estampilles temporelles tout au long de l'enregistrement des événements. Il s'agit d'une approche « échantillonnage avant » (SB). Ceci permet de compenser le décalage initial. Le système de traçage prend également une mesure à la fin de l'exécution. Une application de support peut faire une correction linéaire dans une étape de posttraitement. Ceci permet de compenser le décalage fréquentiel.

Jumpshot

Jumpshot est un outil de visualisation de traces conçu pour les systèmes distribués. Il s'agit exclusivement d'un outil de visualisation, il dépend donc d'un autre programme pour générer les traces. Plusieurs traceurs permettent de convertir des traces de leur format natif à celui de Jumpshot, SLOG-2. L'outil de traçage ayant le lien le plus étroit avec Jumpshot est la bibliothèque de traçage qui est distribuée avec MPE. MPE est un ensemble d'outils liés à la bibliothèque MPI dont Jumpshot fait lui-même partie. MPE est à son tour distribué avec l'implantation de MPI nommée MPICH.

La bibliothèque de traçage de MPE permet de tracer aisément tous les appels à des fonctions MPI lors de l'exécution d'un programme utilisant cette bibliothèque. Pour ce faire, elle tire avantage de l'interface de profilage PMPI, une caractéristique de la spécification MPI qui permet d'intercepter facilement les appels aux fonctions de la bibliothèque (Message Passing Interface Forum, 1995). Afin de donner une estampille temporelle aux événements, MPE peut utiliser la fonction `MPI_Wtime()` qui retourne une valeur de temps. La spécification MPI mentionne toutefois qu'il n'est pas requis que les valeurs de temps retournées par cette fonction soient globales au système distribué, bien que ce soit possible selon l'implantation. Advenant qu'elles ne le soient pas, MPE fournit une option pour synchroniser les horloges.

Lors du traçage d'un programme distribué, MPE enregistre les événements propres à chaque noeud localement durant l'exécution. À la toute fin de celle-ci, l'ensemble des événements est rapatrié, classé et fusionné sur un noeud maître. Lorsque l'option `MPE_CLOCKS_SYNC` est active durant l'exécution d'un programme tracé, MPE va calculer le décalage entre les horloges à la fin de l'exécution et ajouter l'information aux traces, avant qu'elles ne soient envoyées au noeud maître (Mathematics and Computer Science Division - Argonne National Laboratory, 2009). Il s'agit donc d'une approche « échantillonnage après » (SA). Il est aussi possible pour le programme d'appeler manuellement une fonction calculant le décalage entre les horloges en cours d'exécution, à un moment jugé opportun. Cette information est aussi ajoutée à la trace. Les estampilles temporelles sont finalement ajustées en sections linéaires selon chaque mesure du décalage qui a été prise. Ceci est fait lors de l'envoi au noeud maître.

La façon de mesurer le décalage est la même que celle utilisée par VampirTrace. Il est toutefois possible de spécifier certains paramètres dont le nombre d'échanges par ronde. MPE propose également plusieurs façons de propager les valeurs entre les noeuds, plus ou moins précises ou longues à exécuter. Il est par exemple possible d'évaluer le décalage entre l'horloge du noeud maître et de chaque autre noeud individuellement ou alors de manière étagée en arbre pour économiser quelques évaluations.

2.3 Conclusion de la revue de littérature

Face aux possibilités insuffisantes des algorithmes basés sur l'ordre partiel, à l'intrusivité et au manque d'exactitude des algorithmes basés sur la synchronisation en ligne, il semble que les algorithmes de synchronisation hors ligne soient les mieux adaptés à la synchronisation de traces provenant de systèmes distribués.

Les algorithmes de synchronisations hors ligne ont évolué et sont disponibles sous plusieurs variantes. Au niveau de la fidélité, une des deux propriétés présentées à la section 1.1, la plupart des algorithmes visent une bonne précision, mais aucune étude ne compare l'ensemble

des algorithmes l'un par rapport à l'autre dans les mêmes conditions. Ceci s'explique par plusieurs raisons. Certains de ces algorithmes n'ont pas d'implantation connue. Ceux qui en ont sont liés exclusivement à un traceur ou à un type de trace. Afin d'en réaliser une comparaison, il serait nécessaire d'adapter ou de réimplanter plusieurs algorithmes. Ceci représente un travail de grande envergure, ce qui explique sans doute l'absence de ce genre d'étude.

Du point de vue de la justesse, certains des algorithmes présentés offrent des garanties d'ordre partiel. D'un point de vue pratique, certains s'exécutent en temps linéaire. L'algorithme de synchronisation basé sur les enveloppes convexes est le seul qui comporte ces deux propriétés. Deux bémols sont à noter. Il est possible d'avoir des bornes sur la justesse des paramètres de la fonction de conversion de temps, mais pas sur la justesse des conversions elles-mêmes. D'autre part, bien qu'il existe une stratégie d'implantation linéaire de cet algorithme, les implantations connues ont un ordre d'exécution moins bon.

Finalement, le niveau de raffinement des algorithmes présentés contraste avec celui des algorithmes effectivement utilisés dans les outils pratiques de traçage. Ceux-ci se fient majoritairement à des variantes de l'algorithme d'échantillonnage avant et après. De plus, ils sont tous limités à traiter un seul type de trace.

Le prochain chapitre décrit une implantation de l'algorithme basé sur les enveloppes convexes. Ce chapitre décrit aussi une nouvelle variante de l'algorithme qui permet de déterminer la justesse des conversions de temps en tout point d'un groupe de traces.

CHAPITRE 3

ACCURATE OFFLINE SYNCHRONIZATION OF DISTRIBUTED TRACES USING KERNEL-LEVEL EVENTS

Authors

Benjamin Poirier
École Polytechnique de Montréal
benjamin.poirier@polymtl.ca
Robert Roy
École Polytechnique de Montréal
robert.roy@polymtl.ca
Michel Dagenais
École Polytechnique de Montréal
michel.dagenais@polymtl.ca

Submitted to Operating Systems Review (ACM)

Categories and Subject Descriptors

D.2.5 [Software Engineering] Testing and Debugging - tracing
C.2.4 [Computer-Communication Networks] Distributed Systems
G.1.6 [Numerical Analysis] Optimization - linear programming

General Terms Algorithms, Experimentation, Measurement, Performance

Keywords Offline synchronization, time synchronization, trace synchronization, synchronization, timestamp, convex hull, events, kernel

Abstract

Tracing has proven to be a valuable tool for identifying functional and performance problems. In order to use it on distributed nodes, the timestamps in the traces need to be precisely synchronized. The objective of this work is to improve synchronization of traces recorded on distributed nodes. We aim for high precision and low intrusiveness. In this paper, we present an offline trace synchronization algorithm that can report strict bounds on accuracy, an efficient implementation of this algorithm, and an experimental study of parameters that affect synchronization accuracy.

3.1 Introduction

Event tracing has proven its worth. It is used in research and in industry (Bligh *et al.*, 2007). It has helped to identify race conditions, functional problems in IO schedulers, performance problems in device drivers and more. Tracing consists in recording certain events during program execution. These events are made of an identifier, a timestamp and optional parameters. A tracing statement may be seen as a high performance `printf` statement.

The benefits of tracing can be extended to distributed systems: client-server applications, RPC (Remote Procedure Call)-based clusters or HPC (High Performance Computing) applications. A scalable approach is to record a trace individually on each node and to analyze the merged traces afterwards. For this to be meaningful, however, the event timestamps will have to be precisely synchronized across the machines of the distributed system. This is not without its challenges. Tracing can record events with nanosecond precision while network latency is well into the microseconds.

The objective of this work is to improve the synchronization of traces recorded on distributed nodes. The goal is to achieve high precision and low intrusiveness. A preferred approach in this case is to use a system that analyzes events in a post-processing step ("after the fact"). This is called offline synchronization.

The precision of distributed trace synchronization is affected by four factors:

1. communication latency, for example, network latency
2. timestamping latency, the difference in time between the occurrence of an event and its timestamping
3. synchronization algorithm precision
4. communication patterns, for example, the distribution of packets in time

This paper presents a method to improve the second factor and give strict bounds on the third.

In the next section we outline previous work in the area of trace synchronization, especially, various offline synchronization algorithms that have been proposed. We look at the guarantees they provide and their performance. In section 3.3, we present an efficient synchronization algorithm that can report strict accuracy bounds at any point in the trace. We also present the details of an open source implementation that uses kernel-level tracing to reduce timestamping latency. In section 3.4, we report the results of experimentation including long running and large traces. These contain millions of events recorded on real systems in various conditions. We outline factors that influence synchronization accuracy and discuss on actual precision versus guaranteed accuracy and the validity of the assumptions made. We

then conclude by looking back at the significance of what was presented and suggest some theoretical and practical areas to investigate in the future.

3.2 Previous Work

A seemingly simple approach for precise synchronization of distributed systems is to physically distribute a clock to each node. Not only would this require specialized hardware but it would also mean that what was initially a *distributed* system would now be dependent upon a *central* clock. A software solution is desirable.

3.2.1 Clock Model

A preliminary step to the conception and comprehension of synchronization algorithms is the modelization of what we are trying to correct: the inaccuracies in computer clocks. The difference in reading at a time t between a real (physical) clock and a perfect clock can be modelled as (Ellingson et Kulpinski, 1973):

$$\Delta(t) = \alpha(t_0) + \beta(t_0)(t - t_0) + \delta(t - t_0)^2 + \epsilon(t) \tag{3.1}$$

In our case, t_0 can be considered to be the time at which tracing is started. Different sources use different names for each parameter, we will use the following (with other commonly used terms in parenthesis):

$\Delta(t)$	Time offset to a perfect clock
$\alpha(t_0)$	Initial offset
$\beta(t_0)$	Frequency offset (*skew, drift, time offset rate*)
δ	Frequency drift (*drift, drift fluctuation, frequency offset rate, frequency change rate*)
$\epsilon(t)$	Other factors, including random perturbations

Equation 3.1 shows that clock inaccuracies are a compound of different factors. Over relatively short intervals, many algorithms consider that only the initial offset and the frequency offset are significant. We will refer to this as the "linear clock approximation". Under it, equation 3.1 can be simplified as:

$$\Delta(t) = \alpha(t_0) + \beta(t_0)(t - t_0) \tag{3.2}$$

Finding the time offset between a node's clock and a virtual perfect clock becomes a matter of identifying two factors in a linear equation. It follows that the offset between two

real clocks can also be modelled as a linear function. For the rest of this paper, we shall designate an estimate of the function that maps the time on clock A to the time on clock B as:

$$t_{iB} \approx C_B\left(t_A\right) = a_0 + a_1 t_A \tag{3.3}$$

Quartz oscillators (found in commodity computer clocks) have typical frequency drift that results in an error between .3 and 30 ns over 10 minute integration intervals (Ellingson et Kulpinski, 1973). Notably longer intervals can be modelled using a succession of linear functions.

3.2.2 Online Synchronization

An often used method for computer time synchronization is NTP, the network time protocol. NTP is very stable on the long term, however, it is rather jittery on the short term (Mills, 1994, 2006). Moreover, it works by constantly making adjustments to the local clock. In order not to disturb the system that is being measured, one of the objectives of tracing is to be as least intrusive as possible. The use of a system that sends messages and alters the clock during tracing is not desirable. This rules out online synchronization.

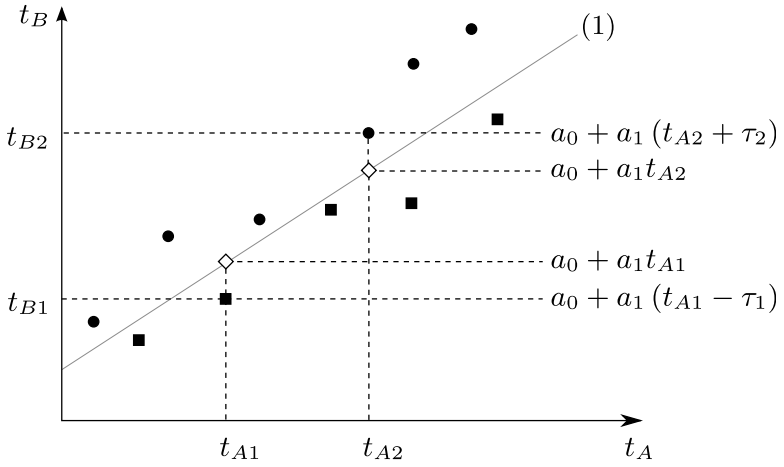
HPC systems, where performance is of the essence, have had tracing for a long time (Browne *et al.*, 1998). The tracing frameworks available are usually centered around the messaging library in use (for instance, the Message Passing Interface, MPI). In the best cases, synchronization is based on rounds of message exchanges (*i*) around (before and/or after) or (*ii*) during the tracing interval (Doleschal *et al.*, 2008). In other cases it is simply left up to the user. Approach (*i*) cannot scale to long tracing intervals (for the reasons outlined in section 3.2.1) and approach (*ii*) is intrusive.

3.2.3 Offline Synchronization

Another solution is to use a system that analyzes events in a post-processing step. The events to look at are those that occur naturally in the traces and that can be linked in multiple traces with a strict ordering relationship. The best example is network packets: a packet is always received after it was sent. From this analysis, it is possible to estimate the difference between the clocks of the traced systems. Effectively, it is possible to synchronize the traces.

The seminal work in this area consists in the linear regression and the convex hull algorithms proposed by Duda *et al.* (1987). These algorithms are used to estimate a conversion function between a pair of traced nodes. They are based on a graphical representation (fig.

3.1) of the messages exchanged between the two nodes. Message i, traced at time t_{Ai} by node A and time t_{Bi} by node B, is represented as the point (t_{Ai}, t_{Bi}). Under the linear clock approximation, the conversion function (which, in practice, is unknown) appears as line (1) on fig. 3.1. Because of positive network latency τ_i (which is also unknown), messages sent from A to B always appear above the conversion function and messages sent from B to A, below. This results in an empty "corridor" around the conversion function.



(1) $C_B(t_A) = a_0 + a_1 t_A$

- messages sent from A to B, $(x, y) =$ (emission timestamp on A, reception timestamp on B)
- messages sent from B to A, $(x, y) =$ (reception timestamp on A, emission timestamp on B)

Figure 3.1 Message representation

Since the conversion function is a line (eq. 3.3), the natural approach to estimate it is to use a linear regression. This is easily implemented with $O(n)$ run time order (in regards to the number of traced events) but it is a fairly weak synchronization. Beyond the question of effective precision achieved (which is reportedly low, Ashton (1995a)), the fact is that this algorithm provides no guarantee against message inversions. It is possible for traces synchronized with such a function to show messages travelling backwards in time. This is clearly an undesirable situation: users of a tracing tool must be able to rely on the data it reports.

In order to avoid message inversions, the estimated conversion function must lie below messages going to B and above messages going to A - it must stay in the empty corridor. There are usually many lines that respect these constraints. The convex hull algorithm identifies the two lines with the minimum and maximum slope. As for the linear regression, it is possible to implement the whole algorithm in $O(n)$ run time order (Haddad, 1988). In

contrast however, this algorithm guarantees no message inversion and provides strict bounds on the values of a_0 and a_1.

Algorithms based on the graphical representation in fig. 3.1 only apply to pairs of systems. A separate factor-propagation step is needed when synchronizing more than two nodes (Duda *et al.*, 1987; Jezequel et Jard, 1996).

Further refinements upon the convex hull algorithm have been proposed. It is possible to adjust the message points according to the minimum message latency for the network in use (Ashton, 1995a). This has the effect of narrowing the empty corridor and improving the accuracy.

Sirdey *et al.* have described a variant targeted towards a specific application where it is sufficient to synchronize the frequency of two systems (Sirdey et Maurice, 2008). This variant is particularly interesting because it uses the same geometric interpretation of the problem as the convex hull algorithm but estimates the conversion function by using a linear programming (LP) approach. The estimate of $\alpha(t_0)$ is used to formulate the objective function and each message point is a constraint. This linear program can be solved in $O(n)$ run time order.

Linear programming has also been used to synchronize traces based on broadcast messages (Scheuermann *et al.*, 2009). This method is based on the observation that a broadcast message sent on a local network will be received by each node at almost the same time: the differential broadcast delay of synchronized traces should be low. A maximum likelihood estimator is used to find correction function estimates for a group of nodes at once. Although this algorithm does not benefit from a strictly linear run time it has been applied to traces containing a total of 10^5 events and 100 nodes. This algorithm does not provide guarantees against message inversions.

3.3 Methodology

The method proposed by Duda to find the boundary values of a_0 and a_1 is illustrated in fig. 3.2. The method considers each message as belonging to one of two sets according to the message direction, M_{AB} for messages going to node B and M_{BA} for messages going to node A. The conversion function estimate has to lie below any message point belonging to M_{AB} and above any belonging to M_{BA} to guarantee no message inversion. The first step is to find the set H_{AB}, the points that form the lower half of the convex hull of the points in M_{AB}, and H_{BA}, the upper half of the convex hull of the points in M_{BA}. The points in each hull are then used to find the conversion function with the maximum slope, $C_B^{max}(t_A) = a_0^{min} + a_1^{max} t_A$, and the one with the minimum slope, $C_B^{min}(t_A) = a_0^{max} + a_1^{min} t_A$. Duda suggests to use the

bisector of the angle formed by these two lines as the estimation of the conversion function, $C_B(t_A)$.

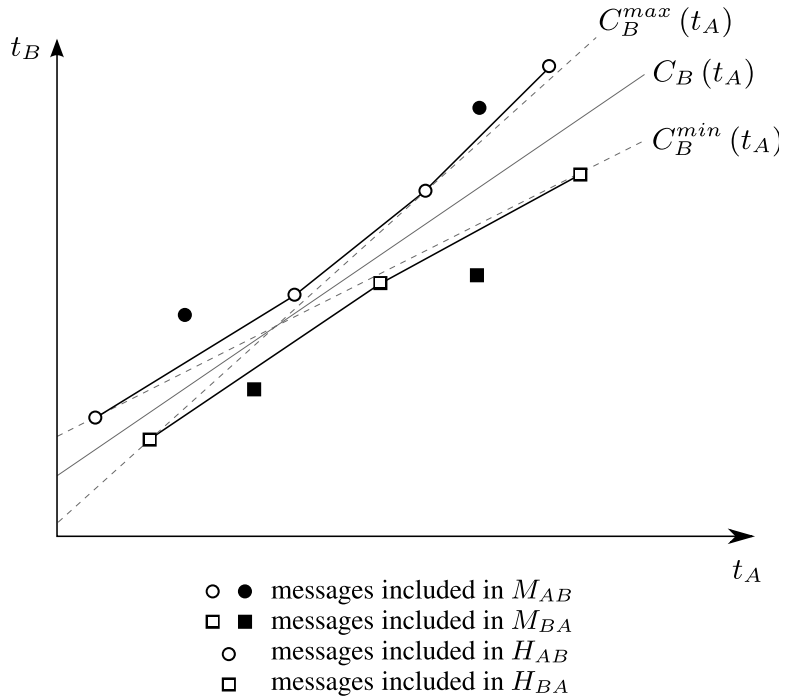


Figure 3.2 Elements of the convex hull method

The half-convex hulls can be formed using Graham's scan algorithm. It is natural to record messages in the trace in such a way that the message points are already sorted. In this case, the time to run the scan is $O(n)$. Each of the two lines can then also be found in linear run time, with respect to the number of points in H_{AB} and H_{BA}, using a specialized algorithm described by Haddad (Haddad, 1988).

3.3.1 Accuracy-Reporting Convex Hull Algorithm

As mentioned in section 3.2.3, the convex hull algorithm guarantees that there will be no message inversion and provides bounds on a_0 and a_1, the parameters of eq. 3.3. Unfortunately, these bounds are not sufficient to provide bounds on a time conversion:

$$t_B \in [C_B(t_A) - \Delta_2 .. C_B(t_A) + \Delta_1] \tag{3.4}$$

It may be tempting to use a_0^{max} and a_1^{max} to evaluate Δ_1. However, using these two parameters in eq. 3.3 would yield a conversion function that does no respect the constraints imposed by the message points. This is because a_0 and a_1 are not independent, fixing one to a certain value restricts the possible range of the other. This is illustrated by line (1) in fig. 3.3.

This conversion function uses parameters within the bounds but it generates many message inversions. Furthermore, the knowledge of $C_B^{min}(t_A)$ and $C_B^{max}(t_A)$ is not sufficient to evaluate Δ_1 at every point. For example, at t_{Am} in fig. 3.3, the conversion function estimate (2) that yields the largest estimate of t_{Bm} is neither $C_B^{min}(t_{Am})$ nor $C_B^{max}(t_{Am})$, it is nevertheless a valid conversion function. Moreover, it can be seen that at this point there are many valid conversion function estimates that yield the same t_{Bm} estimate. The same set of remarks apply to Δ_2. The method that follows builds upon the convex hull algorithm to identify Δ_1 and Δ_2 at any point.

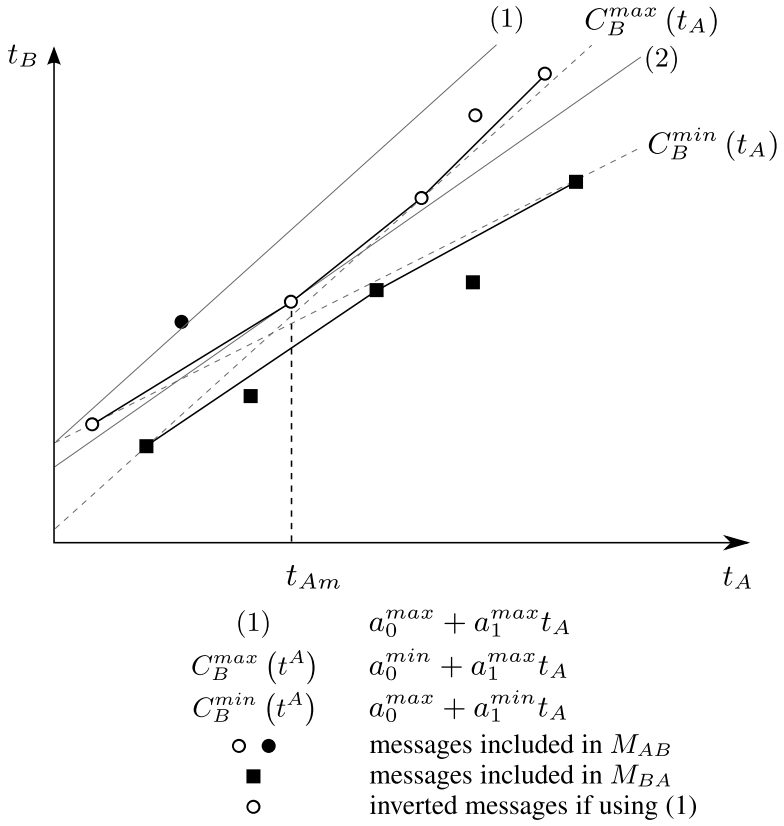


Figure 3.3 Finding a suitable function for accuracy calculation

To begin with, just as Sirdey *et al.* did to estimate $\beta(t_0)$, it is possible to use a linear

program to find $C_B^{max}(t_A)$:

$$\begin{cases} \text{Maximize } a_1 & \\ \text{Subject to} & \\ a_0 + a_1 t_{Ai} \leq t_{Bi}, & (t_{Ai}, t_{Bi}) \in H_{AB} \\ a_0 + a_1 t_{Aj} \geq t_{Bj}, & (t_{Aj}, t_{Bj}) \in H_{BA} \\ a_0 & \text{free} \end{cases} \tag{3.5}$$

Solving this linear program yields a_0^{min} and a_1^{max}. Likewise, the parameters of function $C_B^{min}(t_A)$ can be found by inverting the optimization direction.

Only two variables are involved in this program. This guarantees that it can be solved in linear run time in regards to the number of points in the half-convex hulls (Sirdey et Maurice, 2008). Solving this linear program obviates the need to use Haddad's specialized algorithm.

Linear program 3.5 can be modified to identify the conversion function with the maximum value at a point of interest, t_{An}, rather than the one with the greatest slope. Only the objective function has to be modified, it becomes:

$$\text{Maximize } a_0 + a_1 t_{An} \tag{3.6}$$

We will designate the values of a_0 and a_1 found using this program $l_{0,n}^{max}$ and $l_{1,n}^{max}$. By inverting the optimization direction, $l_{0,n}^{min}$ and $l_{1,n}^{min}$ can be found. From these and $C_B(t_A)$, the estimator suggested by Duda, found using Haddad's algorithm or linear program 3.5, we have:

$$\Delta_1 = l_{0,n}^{max} + l_{1,n}^{max} t_{An} - C_B(t_{An}) \tag{3.7}$$

$$\Delta_2 = C_B(t_{An}) - l_{0,n}^{min} + l_{1,n}^{min} t_{An} \tag{3.8}$$

We are therefore able to directly convert the time with a guaranteed accuracy between the clocks of two traced nodes. The assumptions underlying this result are the linear clock approximation and the availability of sufficiently fine-grained timestamps (Haddad, 1988). In practice, the timestamp resolution on modern computers is much better than the message latency and so the second assumption is always satisfied. Section 3.4.5 will provide experimental data that evaluates the validity of the first assumption.

Algorithmic Complexity

The run time of finding Δ_1 and Δ_2 for one point is composed of the time needed to construct the half-convex hulls and the time needed to solve linear programs 3.5 and 3.6. To repeat the accuracy calculation for more points, it is only necessary to repeat the solving of LP 3.6. We have seen that the time to find the convex hulls is linear in regards to the number of message points and that the time to solve the linear programs is linear in regards to the number of constraints.

Sirdey *et al.* use every message point to generate the constraints of their linear program. This is not necessary. It is sufficient to use the points on the two half-convex hulls to generate the constraints. To their credit, Sirdey *et al.'s* approach avoids having to implement and run a convex hull algorithm. If an LP problem like 3.5 or 3.6 only has to be solved once, either strategy is equivalent in terms of result and run time order. In our case however, LP problem 3.6 has to be solved more often, twice for each point where accuracy information is desired. Therefore, we construct the two half-convex hulls and use only their points to generate the constraints. Section 3.4.6 will show that, in practice, this greatly reduces the number of constraints.

Do we really need to solve LP problem 3.6 for every point in the trace where accuracy information is desired? No. The values of Δ_1 and Δ_2 vary throughout the trace as polygonal curves. Because the objective function and the constraints of LP 3.6 are the same, the vertices of the polygonal curves are a subset of the half-convex hull points. It is therefore possible to calculate accuracy information for every event in the trace by solving LP 3.6 for each half-convex hull point and interpolating linearly everywhere else. The algorithmic complexity of calculating strict accuracy bounds for every messages point can then be expressed as $O\left(n + h^2\right)$ where $n = |M_{AB}| + |M_{BA}|$ and $h = |H_{AB}| + |H_{BA}|$. This is quadratic in the worst case. It is possible to create contrived examples where every message point is part of the half-convex hulls. In practical use however, $h \ll n$ (see tab. 3.6) and so the average run time is effectively linear.

This section showed a method to synchronize distributed traces with guaranteed accuracy. The next section shows a method for improving timestamping latency.

3.3.2 Kernel-Level Event Tracing

In order to perform offline trace synchronization, traces must be recorded during application execution. For scalability, those traces are recorded individually on each node. To synchronize the traces in a post processing step using the algorithms described previously, the traces must contain events that have a strict ordering relationship. In our case, those

events are network packets. In most cases however, in order to be useful, the traces should also contain events related to the operating system or a particular application. Typical MPI tracing tools, for example, will record every call to a function of the MPI library, as well as programmer-defined events.

Goal 1 Record as much useful information as possible about system execution. This facilitates analysis of application behavior (core kernel, driver or user-level code).

One primary concern of a tracing tool is to have a low overhead. Tracing should not modify the application behavior or else some problems, such as race conditions, might not be reproducible. Sometimes error conditions only show up after extended periods or under heavy load. It should be possible to use tracing in a "production" environment.

Goal 2 Reduce the computational demand of tracing and therefore reduce its intrusiveness.

These goals are applicable to tracing in general. Trace synchronization brings extra concerns. One of the key components influencing synchronization accuracy is the width of the empty corridor on fig. 3.2. This width is modulated in part by the timestamping delay: the lapse of time between (*i*) the timestamping of a message and its emission and (*ii*) the arrival of a message and its timestamping. These latencies reduce the precision of the timestamp and in turn reduce the accuracy of the synchronization.

Goal 3 Timestamp a packet as late as possible before its emission and as soon as possible after its reception.

The timestamping delay will vary according to the point at which the event is generated. Various choices are available. Generally, the journey of data to be encapsulated into a packet and emitted onto the network is as follows: from user-level application code to a support library, handed to the kernel, through the network stack to the network interface card (NIC) driver and finally to the NIC hardware.

Goal 4 Do not require specific hardware or modifications to the traced application.

Supporting only a type of application or a group of networking cards would severely limit the applicability of a tracing framework.

In light of these goals, we chose to use the Linux Trace Toolkit Next Generation (LTTng) (Desnoyers, 2009). This tracer operates mainly at the kernel-level and records events related to many parts of the operating system (OS) (process scheduling, memory management, file system operations, interrupt handling, ...) This yields information on much of the internals of a system but also on the state of an application and its interaction with the OS. The LTTng tracer also supports inserting custom new event generation statements ("tracepoints") at the kernel and the application level. LTTng uses the timestamp counter (TSC) processor register as a time source. It is precise and fast to read. It is also unaffected by NTP time correction.

Using such a tracer combines the broad capabilities of custom application tracing code, OS tracing, system-call reporting, network traffic analysis and more while reducing tracing framework code duplication.

Does this flexibility impact performance negatively? The LTTng tracer keeps a low overhead by being based mostly on static tracepoints, inserted in the code before compilation. It also uses in-place code modification and efficient synchronization primitives. Average kernel-level events are benchmarked at 119ns on an Intel Core2 Xeon 2GHz (Desnoyers, 2009).

LTTng's support for network tracing had to be extended with new tracepoints to suit the needs of offline synchronization. Goal 3 alone dictates placing the network tracepoints at a level as low as possible. Certain network cards support hardware timestamping. Although using this feature would reduce timestamping latency, it would also go against goal 4. The next level up is to timestamp a packet in the NIC driver. However, this would require modifying every driver, an unwieldy task, or supporting only certain drivers, which also goes against goal 4.

In order to conciliate the third and fourth goals, we generate events at the lowest possible point in the network stack, just before the interface with the NIC driver. Using kernel-level event tracing allows us to timestamp a packet emission after data is handed from an application to the operating system and after it has been processed by the networking stack. This reduces the timestamping delay compared to recording messages at the application level and it means that every combination of application and hardware is supported. Linux *libpcap*-based traffic analyzers use the same tracing point (McCanne et Jacobson, 1993). Since trace events are recorded locally, there is no need to modify the packets. This contributes to keep the intrusiveness of tracing low.

Further conciliation was necessary, between goals 1 and 2. Now that we have chosen *where* to record events, we have to choose *what* to record. An area of practical concern that is often glossed over in the description of synchronization algorithms is how to identify that a transmission event and a reception event in two unsynchronized traces correspond to the same packet (Scheuermann et Kiess, 2009). Recording entire packets provides as much information as is available. However, this can amount to recording the entire stream of data of a NIC operating at line speed, a task that no common hard drive (because of transfer rate) and no in-memory buffer (because of size) can sustain for a long time. It is too intrusive. We sought to reduce the quantity of information recorded about each packet to the minimum needed to support event matching.

The approach taken to match events is to record and compare a total of 31 bytes out of the TCP/IP headers. IP address and port fields uniquely identify a connection while TCP sequence numbers and flags provide a high probability of uniquely identifying a segment

within the connection. A consequence of this is that synchronization is limited to using TCP segments. We believe that this is an acceptable tradeoff, considering the ubiquity of this protocol. If recording distributed traces between nodes that do not exchange TCP traffic, a user may run a simple application that generates some extra traffic. Section 3.4.3 will show that the synchronization can work with a wide range of packet rates.

The LTTng project also includes a trace viewer, LTTV. This program can show graphical views of the state of every process and resource on the system throughout the trace. It can also filter and display raw event listings. The accuracy-reporting synchronization algorithm described in section 3.3.1 has been implemented and contributed to LTTV. The linear programming problems are solved using an open source LP solver, glpk (Makhorin, 2009). This implementation of the synchronization algorithm has been used to perform the experimentation in the next section.

Figure 3.4 is a screenshot of two synchronized traces in LTTV. Trace 0 was recorded on a machine where *wget*, a simple command line web client, was run. Trace 1 was recorded on another machine running the *Apache* HTTP server. This view shows the state of each process as the client initializes and makes its request to the server. One `apache2` process receives the request and dispatches it to a worker thread before the data is sent back to the client which writes it to disk and terminates. It is possible to analyze the state of each process and operating system as well as the timing of every event in a single view.

3.4 Real World Experiments

The following section shows the results of running our synchronization algorithm on groups of traces recorded on a pair of systems on a local network. Each node is equipped with dual quad core Intel E5405 processors at 2.00GHz, 8GB of main memory and can be accessed via two network interfaces: a 100Mbps Fast Ethernet interface and a Gigabit interface. During the experiments, a traffic generator was used to send messages at regular intervals between the nodes. One message corresponds to a pair of related send and receive events. In our case, one message is a single TCP segment sent between two nodes. Our results are presented in terms of messages per second because the algorithm is agnostic with regards to the type of events that generate messages.

3.4.1 Synchronization Evaluation

Most of the following experiments evaluate the influence of different factors on synchronization precision and accuracy. While our algorithm can report accuracy, the real time offset between the two nodes is unknown. We do not know the true value of what we are trying

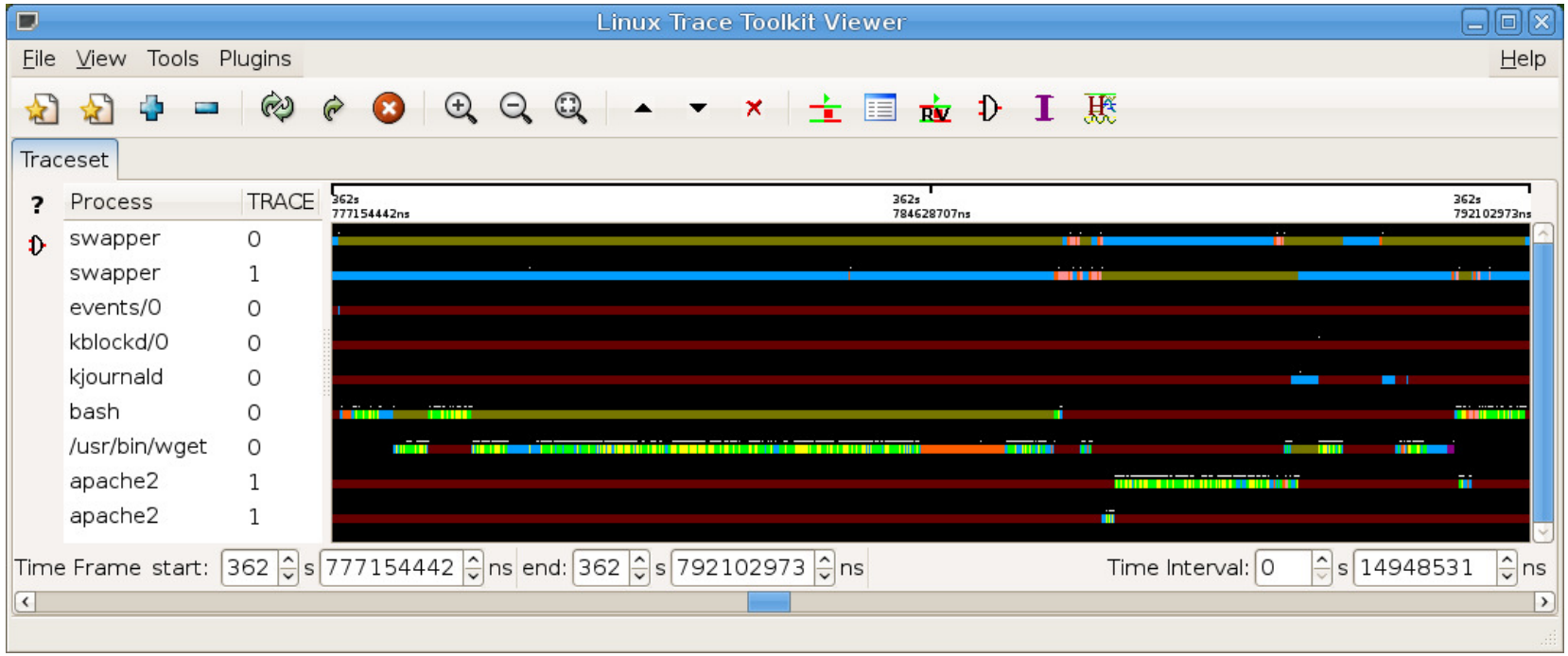


Figure 3.4 LTTV displaying traces from a web client and server

to estimate. This is a common problem when evaluating synchronization algorithms. Some authors use simulated traces. This is an indisputable way of comparing the estimated value against a known reference. However, the truthfulness of the simulation becomes an extra concern. Since the algorithm will ultimately be applied to real traces, we sought to perform our experimentation using real traces as well. Others have also followed this path. This is the case of Ashton who proposed some metrics to compare different synchronization algorithms (Ashton, 1995a).

We followed the same approach and chose three types of metrics to evaluate the synchronization precision. These metrics have the bonus that they can also be helpful to users of the algorithm in a practical context. The first type of metrics is based on messages. We reuse two of Asthon's metrics: message inversions and messages running too fast. The calculations are performed on the synchronized trace set. In the first case, the number of packets that appear to be received before they are sent is identified. In the second case, the number of packets that exhibit a message latency smaller than the minimum network delay is identified.

The number of message inversions will always be null when using the convex hull algorithm. This is a guarantee it provides.[1] The number of messages running too fast on the other hand should be as small as possible. To evaluate this metric the minimum network delay, τ_{min}, has to be known. Several tools can estimate the minimum Round Trip Time (RTT) between two nodes. We considered the network to be symmetric, therefore $\tau_{min} = {}^1\!/_2 \mathrm{RTT}_{min}$. We compared the output of `ping`, which takes a measure at the ICMP level, `netperf` running a test at the UDP level and `nuttcp` running a test at the TCP level. The results obtained were consistently increasing in that order, which is also the order of increasing protocol complexity. As each of the nodes was using two network interfaces, the tests were repeated between each address of each node so as to use the value appropriate to each packet. Measures were also taken in both directions, exchanging the node initiating the command or the role of client and server were applicable. The τ_{min} values obtained are presented in table 3.1. We chose to use the ICMP values to perform our measures, they should be the ones closer to the "true" minimum network delay.

The second type of metrics was inspired by the synchronization algorithm based on broadcast messages described at the end of section 3.2.3. An extra node was used to send UDP broadcast datagrams at the same interval as the TCP segments. Events were recorded for those messages in the traces. They were not used to perform the synchronization but only as an independent indication of its precision. Once the traces are synchronized, the difference in apparent arrival time of the same broadcast on each node is measured. The better the

1. If there are no linear conversion functions that can provide this guarantee, a practical tracing tool should be able to fallback to a best efforts approximation. Section 3.5 explores this topic.

Table 3.1 Minimum Network Delay (ms) (n0: node 0, n1: node 1, FE: 100Mbps Ethernet, GE: 1Gbps Ethernet)

Source	Destination	ICMP	UDP	TCP
n0-FE	n1-FE	0.103	0.125	0.180
n1-FE	n0-FE	0.084	0.131	0.180
n0-GE	n1-GE	0.023	0.053	0.055
n1-GE	n0-GE	0.031	0.054	0.055

synchronization of the traces is, the smaller the broadcast differential delays should be.

The third type of metrics is based on the accuracy reported by our algorithm. We find the best, worst and average accuracy range throughout the trace duration.

3.4.2 Variation of Network Type

The first group of traces was recorded for a duration of 120 seconds during which UDP and TCP packets were exchanged at the rate of one per second on the Fast Ethernet interfaces.

The algorithm first identifies $C_B(t_A)$, the clock conversion function estimate. Because of the scale of the network message latency compared to the duration of the traces, the convex hulls are not discernible at a scale suitable for print when using a representation similar to fig. 3.1. Instead, fig. 3.5 shows a zoomed-in view of the area around the very first message points. This figure contains TCP message points, half-convex hull outlines, conversion function estimate and synchronization accuracy. More than one message point appears in an interval of less than a second. This is because each node is sending TCP segments at the rate of one per second and because TCP acknowledgments are sent automatically in response to data.

The novelty of our algorithm is to be able to calculate accuracy bounds at any point in the trace. In the area shown on fig. 3.5 the accuracy bounds correspond to the minimum and maximum conversion function estimates. This is not the case throughout the trace duration. Figure 3.6 uses a different representation to show the accuracy bounds calculated by our algorithm as well as the differential delay of each UDP broadcast for the entire trace duration. The graph can be interpreted as follows:

- at time $t_A = 100\,\mathrm{s}$ for example,
  $t_B \in [C_B(t_A) - 3.56 \times 10^{-5}..C_B(t_A) + 3.41 \times 10^{-5}]$ so, at $t_A = 100\,\mathrm{s}$ on fig. 3.6, the "Synchronization accuracy" area extends from -3.56×10^{-5} to 3.41×10^{-5}
- a broadcast timestamped at time $t_A = 85.8\,\mathrm{s}$ was timestamped with a clock value $2.41 \times 10^{-5}\,\mathrm{s}$ smaller on node B so a "Broadcast differential delay" point appears at $(85.8, -2.41 \times 10^{-5})$

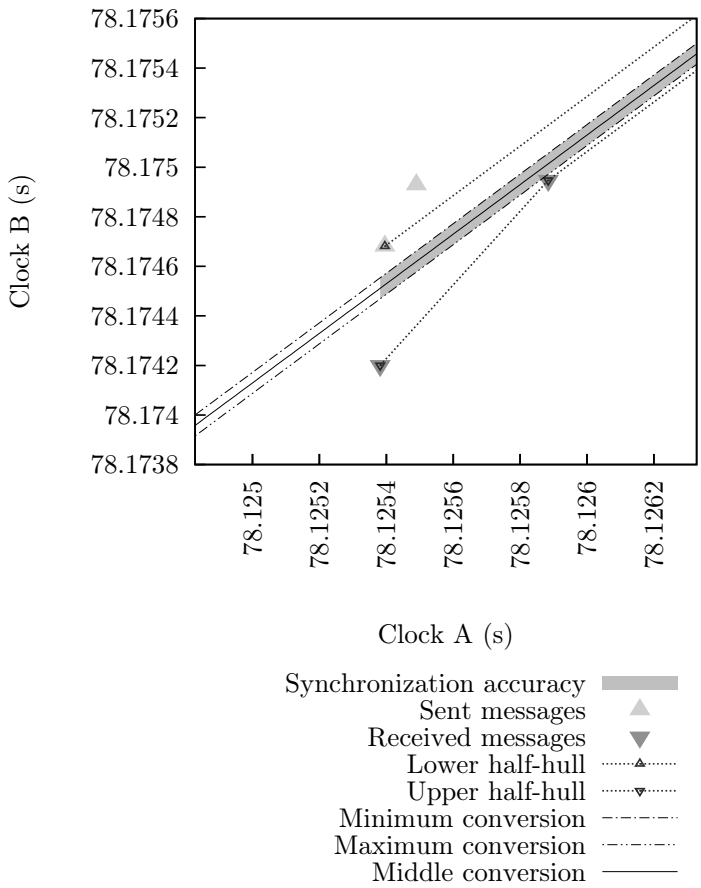


Figure 3.5 Conversion function (detail); 120 s, 1 $^{\text{msg}}/_{\text{s}}$, Fast Ethernet

We recorded a second pair of traces in the same conditions except that the traffic circulated on the Gigabit Ethernet network. Figure 3.7 shows the accuracy bounds and broadcast differential delays for this case. The peculiar diagonal pattern apparent on figures 3.6 and 3.7 is caused by an adaptive interrupt moderation algorithm associated with the NICs in use, Intel PRO/1000 (Prasad *et al.*, 2004). The synchronization evaluation metrics for the Fast Ethernet and Gigabit Ethernet traces are reported in table 3.2. As expected, there are no message inversion.

We may first notice that there is a notable asymmetry apparent in the trace group recorded on Gigabit Ethernet. Looking at the message metrics in table 3.2, messages seemed to be travelling faster than the network's minimum delay in only one direction, to node B. This condition suggests that the synchronized timestamps on node B are too early. Looking at the broadcast differential delay metrics, these indicate that broadcasts were, on average, received by node B 1.26×10^{-5} s before node A. Once again, the timestamps recorded by node B were "too small". The two metrics are in agreement. This does not imply an asymmetric network

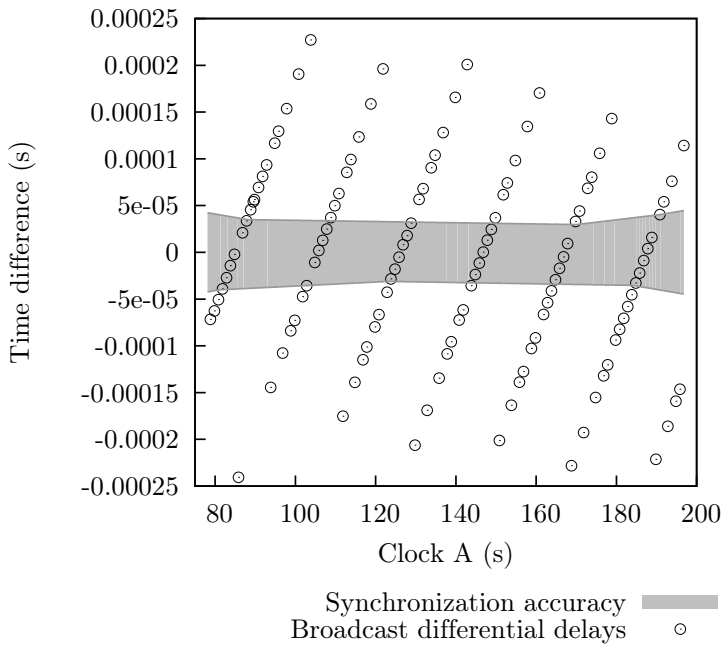


Figure 3.6 Synchronization accuracy; 120 s, 1 $^{msg}/_{s}$, Fast Ethernet

however. As Haddad pointed out (Haddad, 1988):

"Considering two nodes with linear clocks in a distributed system, it is impossible to tell the difference between clock offset and minimum network delay asymmetry. This uncertainty is bounded by the sum of the minimum network delay in each direction."

The average broadcast differential delay, which is an indication of clock offset after synchronization, is indeed smaller than the sum of the minimum network delay in each direction as found in table 3.1.

We may then notice, by looking at fig. 3.6 and 3.7, that the spread of broadcast differential delay values is smaller on Gigabit Ethernet. Table 3.2 confirms this. Delay range and standard deviation are smaller, which suggests a lower network jitter.

Table 3.1 shows that the Gigabit network has a lower latency. This has the effect of reducing the width of the empty corridor on fig. 3.1 and therefore improving the accuracy. Indeed, all three accuracy metrics in table 3.2 are better in the case of the Gigabit network.

We may notice in all accuracy graphs that many of the broadcast differential delay values are outside of the strict accuracy bounds reported by the algorithm. While this may seem contradictory at first, the cause is that synchronization precision is only one factor affecting broadcast differential delays. These delays are also modulated by the network latency and timestamping latency.

Finally, looking once again at the average broadcast differential delay, we notice that it is

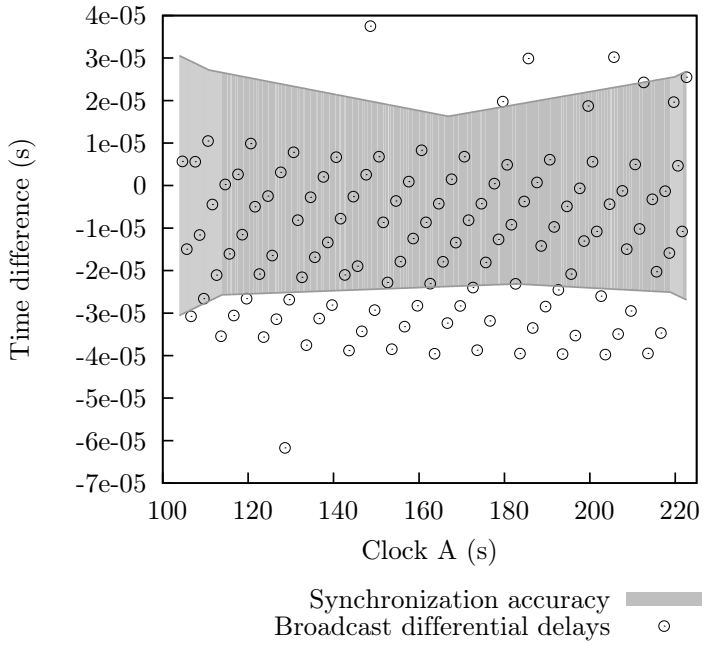


Figure 3.7 Synchronization accuracy; 120 s, 1 $^{msg}/_{s}$, Gigabit Ethernet

smaller in the case of Fast Ethernet. This suggests that synchronization precision is better with the trace set recorded on Fast Ethernet than with the trace set recorded on Gigabit Ethernet. On the other hand, the overall number of messages running too fast is smaller with Gigabit Ethernet. This suggests the precision is better on the Gigabit network. This time, the two metrics disagree. Upon repeating this experiment we did notice that the number of messages running too fast was consistent whereas the average broadcast differential delay had a large variance. The unexplained variance of the later suggests that there is another parameter, which we did not control, that has a measurable impact on trace synchronization precision, perhaps clock non-linearity.

In summary, the message metrics indicate that synchronization precision is better on the Gigabit network and the accuracy metrics indicate that accuracy is also better on this network. This network has a lower latency and jitter but shows some asymmetry in the synchronized traces. The metrics based on messages are more consistent than those based on broadcast differential delay.

3.4.3 Variation of Message Rate

We repeated the experiment, this time by using Gigabit Ethernet exclusively but increasing the rate at which TCP and UDP packets were sent to 16 messages per second. Compared to the previous case with 1 $^{msg}/_{s}$, the accuracy area was again reduced, to slightly better than

Table 3.2 Synchronization evaluation metrics, variation of network type

	120 s, 1 msg/s, Fast Ethernet	120 s, 1 msg/s, Gigabit Ethernet
Messages		
Inversion	0	0
Too Fast (A to B)	11.3%	13.9%
Too Fast (B to A)	14.3%	0.0%
Too Fast (Overall)	12.8%	6.9%
Broadcast Diff. Delay		
Minimum (s)	-2.40×10^{-4}	-6.17×10^{-5}
Maximum (s)	2.27×10^{-4}	3.75×10^{-5}
Average (s)	-1.08×10^{-5}	-1.26×10^{-5}
Standard Deviation (s)	1.03×10^{-4}	1.81×10^{-5}
Accuracy ($\Delta_1 - \Delta_2$)		
Best (s)	6.39×10^{-5}	4.00×10^{-5}
Worst (s)	8.93×10^{-5}	6.11×10^{-5}
Average (s)	6.88×10^{-5}	4.66×10^{-5}

an almost constant ±20 us. In order to seek out the best accuracy achievable, the message rate was further increased in an exponential fashion by doubling it successively up to 1024 messages per second. The total tracing duration was kept constant at 120 s. The results are presented in table 3.3.

Table 3.3 Synchronization evaluation metrics, variation of message rate

Rate (msg/s)	Messages Running Too Fast	Broadcast Differential Delay Average (s)	Accuracy Average (s)
16	7.09%	-7.17×10^{-6}	3.74×10^{-5}
32	6.44%	-8.76×10^{-6}	3.69×10^{-5}
64	6.23%	-9.31×10^{-6}	3.53×10^{-5}
128	6.12%	-9.03×10^{-6}	3.46×10^{-5}
256	6.28%	-9.02×10^{-6}	3.43×10^{-5}
512	6.14%	-8.43×10^{-6}	3.19×10^{-5}
1024	5.86%	-8.57×10^{-6}	3.07×10^{-5}

As can be seen in table 3.3, increasing the message rate improves the average accuracy bounds. We theorize that this is because increasing the overall number of messages transmitted increases the number of messages sent and received with a latency close to τ_{min}. This has the effect of narrowing the empty corridor between the two half-convex hulls. This, in turn, has the effect of improving the accuracy. The number of messages with a higher latency will

also increase. However, because the convex hulls will be formed predominantly with messages of low latency, this does not have a negative impact on accuracy.

As was the case in section 3.4.2, the broadcast differential delay averages do not show a clear trend. Once again, the percentage of messages running too fast is a more consistent metric. It shows that as the message rate increases, so does the precision.

In summary, increasing the message rate improves the accuracy and the precision. This was theorized by Duda which stated that, as the number of messages increases, $a_1^{max} - a_1^{min} \to 0$ and $a_0^{max} - a_0^{min} \to 2\tau_{min}$.

3.4.4 Variation of Trace Duration

As the previous experiments have shown, increasing the number of messages through an increase in message rate improves the accuracy and precision. Do we get the same benefits if we increase the number of messages through an increase in trace duration? We repeated the experiment in the following conditions: Gigabit network, keeping a fixed 1024 $^{msg}/_{s}$ rate and trace duration doubling successively from 30 s to 960 s. The results are presented in table 3.4.

Table 3.4 Synchronization evaluation metrics, variation of trace duration

Duration (s)	Messages Running Too Fast	Broadcast Differential Delay Average (s)	Accuracy Average (s)
30	5.64%	-9.95×10^{-6}	3.29×10^{-5}
60	5.96%	-8.97×10^{-6}	3.19×10^{-5}
120	6.14%	-9.37×10^{-6}	3.14×10^{-5}
240	6.13%	-8.88×10^{-6}	3.07×10^{-5}
480	6.01%	-1.05×10^{-5}	2.96×10^{-5}
960	6.93%	-1.37×10^{-5}	2.48×10^{-5}

The number of messages increases with the trace duration. As was the case in section 3.4.3, increasing the overall number of messages improves the accuracy bounds. However, in this case, it is misleading. Indeed, the number of messages running too fast also increases, which indicates a loss in precision. The broadcast differential delay averages also point towards a loss in precision. This is because the longer the trace duration, the farther we stray from the linear clock approximation. Since the convex hull algorithm depends on this assumption, it is expected that its results are biased when the assumption is violated.

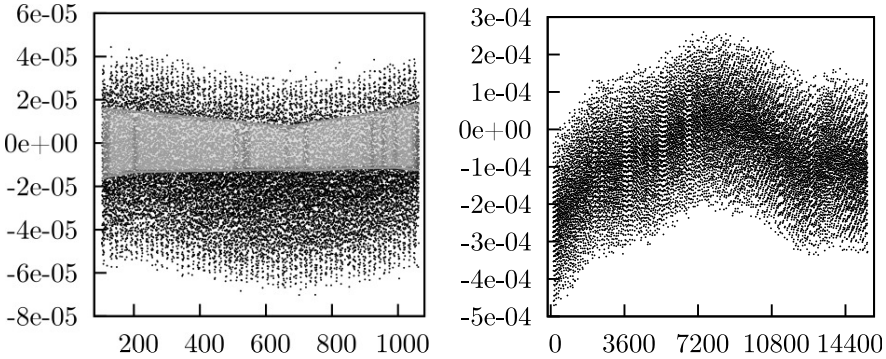
In summary, as the trace duration lengthens, the increasing importance of clock frequency drift causes the hulls to get closer and artificially improves accuracy. It is possible to detect

this by looking at the metrics that reflect synchronization precision.

3.4.5 Long Trace Duration

In the previous section, we have seen that the error committed by using the convex hull algorithm grows as the duration of the trace increases. With some algorithms, like linear regression-based ones, the precision of the estimation will keep on degrading as the non-linear components of the clock equation become more important. With convex hull-based algorithms however, a situation will be reached where it is impossible to produce an estimation. This will happen when the two half-convex hulls intersect each other.

Figure 3.8a shows the accuracy bounds calculated by our algorithm and the broadcast differential delays during the 960 seconds trace duration from section 3.4.4. With careful observation, we can already discern a non-linearity in the broadcast differential delays. Nevertheless, the convex hull algorithm can complete. In contrast, fig. 3.8b shows the broadcast differential delays for a pair of traces running for 15360 seconds (4:16 hours). The trace was recorded on the Fast Ethernet network, with a message rate of $1\,^{\text{msg}}/_{\text{s}}$. In this case, the non-linearity of the clocks causes the two half-hulls to intersect each other. It is not possible to fit a linear function in between them. Thus, we cannot use the formal definition of the convex hull algorithm.



(a) 960 seconds (b) 15360 seconds

Figure 3.8 Long trace durations

With this situation in mind, Ashton suggests a "fallback" mode, derived from the convex hull algorithm. We added this mode to our implementation, as well as a variation of Duda's linear regression algorithm (Clément et Dagenais, 2009). Neither of these two algorithms provides a guarantee against message inversions. Table 3.5 shows the synchronization evaluation metrics for these two modes.

Table 3.5 Synchronization evaluation metrics, long trace duration

	Fallback mode	Linear regression
Messages		
Inversion	4.06%	9.57%
Too Fast	14.17%	18.47%
Broadcast Diff. Delay		
Minimum (s)	-4.70×10^{-4}	-5.05×10^{-4}
Maximum (s)	2.60×10^{-4}	1.91×10^{-4}
Average (s)	-6.72×10^{-5}	1.38×10^{-4}
Standard Deviation (s)	1.20×10^{-4}	1.18×10^{-4}

The broadcast differential delay ranges are about the same regardless of the synchronization algorithm. This is also the case for the standard deviation. This strengthens the point that those metrics reflect characteristics of the network rather than the synchronization. Compare them to table 3.2 for the Fast Ethernet case.

Looking at the messages metrics, this trace group stands out as the only one that presents message inversions after synchronization. The percentage of inversions and of messages running too fast favors the fallback mode of the convex hull algorithm. This is confirmed by the broadcast differential delay average which is lower.

In summary, the convex hull algorithm cannot be used as-is when the non-linear components of the clocks are important. Section 3.4.4 showed that this first manifests itself as a reduction of precision coupled with an artificial increase in accuracy. This section showed that the convex hull algorithm will fail after a certain point. It is possible to use some alternative algorithms with varying degrees of precision. Others have also suggested to segment the traces in sub-intervals of limited duration.

3.4.6 Algorithmic Performance

We verified experimentally the run time characteristics of our convex hull implementation, needed to find the clock correction factors, and the subsequent step needed to report the accuracy.

We tested the run time of our algorithm on groups of traces including those collected in the previous experiments. The results are presented on fig. 3.9. Note that the figure is in logarithmic scale. The measures were taken on the same type of machine used to record the traces. The algorithm implementation is single threaded.

The sample points for the convex hull analysis in fig. 3.9 confirm that our implementation scales linearly with the number of network events in the traces, even with large traces. We

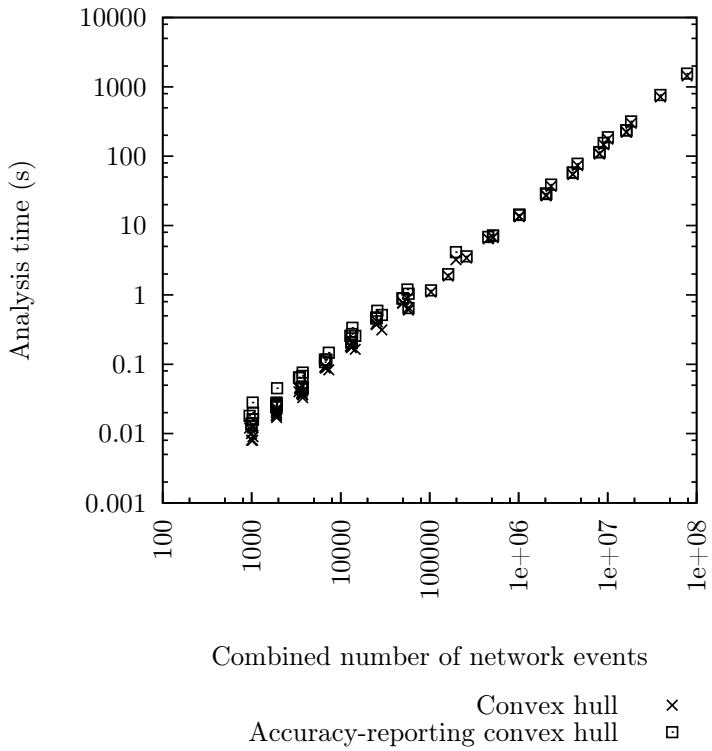


Figure 3.9 Analysis time for synchronization

should expect nothing less, given that a linear time implementation strategy is available. The different analysis times for the same number of network events are due to the fact that traces with different traffic patterns were used.

The synchronization algorithm itself can be applied to any type of trace that includes events happening in distinct traces with a strict ordering relationship. To distinguish between the time consumed by the tracing framework from the time needed by the synchronization algorithm, the measures in fig. 3.9 only include the latter. In practice, some extra time is needed to read the events out of the trace files on disk. This is dependent on the tracing framework in use but can also be implemented in linear time, as is the case for LTTV.

Figure 3.9 also shows the time needed to calculate the synchronization accuracy information needed to generate figures like 3.6. As stated in section 3.3.1, the run time complexity of our algorithm is $O(n + h^2)$. Although it is quadratic in terms of h, the number of convex hull points, we expected that the actual run time would be dominated by the linear part, dependent on n, the number of message points. In practice, even when n increases to millions, h stays well under 100. This is shown in table 3.6. The actual run time of the accuracy reporting synchronization algorithm is shown in fig. 3.9 and confirms our expectations, it closely follows that of the regular convex hull algorithm.

Table 3.6 Number of points in convex hulls

Message points	Convex hull points	Ratio
7690	25	0.325%
14295	23	0.161%
26926	20	0.074%
54019	20	0.037%
108530	28	0.026%
215714	31	0.014%
430429	32	0.007%
860032	26	0.003%
1718787	33	0.002%
3441245	46	0.001%

3.5 Conclusion

It is quite remarkable that any unmodified application exchanging TCP traffic on any hardware is sufficient to perform trace synchronization. The accuracy reporting convex hull algorithm performs offline synchronization of distributed traces. With a linear clock, it guarantees the absence message inversion and gives strict bounds on accuracy at any point in the trace. Its run time is quadratic in the worse case but it scales almost linearly on practical traces.

We have described how to record network events at the kernel level with low intrusiveness. We have also studied practical factors that affect offline synchronization. Accuracy is improved by using a network with lower latency and by using a higher message rate. We have shown experimentally the effects of the linear clock assumption. With a constant message rate, lengthening the trace duration reduces precision and gives a false impression of improving accuracy. This is detected using metrics based on messages running too fast and broadcast differential delays. During our experiments, we have achieved a synchronization accuracy of ± 15 us and an estimated precision of 9 us on a network with an estimated minimum latency of 39 us.

An accuracy-reporting synchronization algorithm can be used to study parameters that affect synchronization. It is also essential to validate the partial ordering of events when causality alone is not sufficient to do so. This is needed to confirm the observations made by users of a tracing tool. It can also be used to confirm assumptions from automated analysis tools.

Our algorithm could be extended to long running and streaming traces by considering sub-intervals. It could also be extended to systems of more than two nodes. Algorithms to propagate the factors efficiently while adjusting the accuracy bounds are needed to do this.

From an applicability perspective, the synchronization algorithm here described can be extended to other types of traces that include events with a strict ordering relationship recorded by distinct clocks. This can be the case of systems with different network types, user-level traces of certain applications or network packet captures.

From a practical perspective, some analysis tools have been developed to analyze the interactions of processes within a single system. These could be extended to work over distributed systems by integrating the accuracy information made available by the accuracy-reporting convex hull algorithm.

CHAPITRE 4

DISCUSSION COMPLÉMENTAIRE

Ce chapitre revient sur les résultats présentés dans l'article du chapitre 3. Certains aspects de ces résultats sont analysés plus en détail en fonction de la revue de littérature étendue du chapitre 2. De plus, quelques contradictions apparentes sont soulevées dans la littérature ainsi qu'entre les résultats expérimentaux obtenus et ceux présentés dans la littérature. Finalement, l'implantation logicielle réalisée est brièvement analysée par rapport aux caractéristiques des implantations rapportées dans la revue de littérature.

4.1 Temps d'exécution de l'algorithme de synchronisation basé sur la méthode des enveloppes convexes

On remarque une certaine confusion dans la littérature quant à l'ordre de complexité de l'algorithme de synchronisation basé sur la méthode des enveloppes convexes. Dans leur article publié en 1987, Duda et ses coauteurs (incluant Y. Haddad) recommandent d'utiliser une marche de Jarvis pour construire les enveloppes convexes (Duda *et al.*, 1987). L'utilisation de cet algorithme mène nécessairement à une complexité $O(n^2)$ en pire cas. Dans sa thèse, publiée en 1988, Haddad recommande plutôt l'utilisation d'un algorithme qui s'avère être un parcours de Graham. Bien que celui-ci comporte une complexité $O(n \log n)$, Haddad précise que, dans le cas de la synchronisation de trace, celle-ci est réduite à $O(n)$ puisque les événements sont déjà triés selon leur estampille temporelle. Haddad décrit par la suite comment réaliser l'ensemble de l'algorithme de synchronisation avec un ordre linéaire. Il est fort possible que cette amélioration de $O(n^2)$ à $O(n)$ ait été développée entre les deux publications. Toutefois, dans sa publication de 1995, Ashton indique que la recherche des enveloppes convexes comporte une complexité $O(n \log n)$ ce qui borne la complexité de l'ensemble de l'algorithme (Ashton, 1995a). De plus, Ashton mentionne que la complexité de l'implantation qu'il a réalisée est quadratique. Bien qu'il fasse référence à l'article publié par Duda et al., Ashton n'était donc probablement pas au courant des travaux de Haddad. Encore aujourd'hui, la thèse de Haddad n'est pas disponible en ligne. De surcroît, celle-ci fut rédigée en français ce qui en réduit la portée.

L'article reproduit au chapitre 3 fait mention des travaux de Haddad. Il indique aussi une deuxième stratégie d'implantation ayant un ordre $O(n)$, dérivée des travaux de Sirdey et Maurice (2008). Finalement, les résultats expérimentaux de la figure 3.9 montrent qu'une

implantation en temps linéaire a été réalisée.

4.2 Durée optimale de la période de synchronisation

Lors de son étude sur la stabilité des oscillateurs au quartz présents dans les ordinateurs, Mills indique qu'un intervalle d'environ 1000 secondes semble être le meilleur compromis pour compenser à la fois les instabilités de courte durée et de longue durée (voir la section 2.2.1). Selon l'expérimentation décrite à la section 3.4.4 et présentée dans le tableau 3.4, il semble toutefois que toute augmentation de la durée de traçage mène à une dégradation de la précision de la synchronisation. Une période de synchronisation de 60 ou de 500 secondes est donc meilleure qu'une période de 1000 secondes. Une explication possible de cette disparité est que les techniques utilisées pour effectuer la synchronisation sont différentes dans les deux cas.

Les travaux de Mills concernent la synchronisation en ligne avec NTP. Ils se basent sur une horloge contrôlée par une boucle de rétroaction de type PLL. Bien que la période d'intégration du signal utilisé en entrée de cette boucle puisse effectivement être optimale à 1000 s, la phase de l'horloge est corrigée beaucoup plus fréquemment dans le système NTP (à chaque seconde ou à chaque interruption de l'horloge d'ordonnancement selon les fonctionnalités offertes par le système d'exploitation (Mills, 2006)). À l'opposé, dans le cadre de nos travaux qui concernent la synchronisation hors ligne, les corrections de temps sont effectuées sur la même période que celle utilisée pour faire les mesures.

L'effet est donc le suivant. Bien que les mesures puissent être faites sur une période de 1000 s dans les deux cas, la synchronisation effectuée par Mills va appliquer une correction non linéaire à l'horloge alors que les algorithmes de synchronisation utilisant la première approximation du modèle d'horloge (tel que la méthode basée sur les enveloppes convexes) vont appliquer une correction linéaire à l'horloge. Au-delà de la période du bruit, l'erreur commise en utilisant la première approximation augmente systématiquement lorsque l'on allonge la durée. Il est donc normal qu'une période de synchronisation plus longue se solde par une précision moindre.

4.3 Comparaison avec les résultats publiés par d'autres auteurs

Parmi les expérimentations que l'on retrouve dans la littérature, le travail se rapprochant le plus du nôtre a été réalisé par Ashton (1995a). Cet auteur a implanté l'algorithme basé sur les enveloppes convexes et a réalisé différentes expériences impliquant des traces enregistrées sur un réseau local. Les traces qui mettent en jeu le plus grand nombre de messages ont été prises durant l'exécution de MUSBUS, un test de performance qui simule une utilisation

« moyenne » d'une station de travail UNIX par un usager. Cette expérience qui implique 5 noeuds a été répétée 10 fois, pour un total de 100 paires de traces à synchroniser. Les résultats moyens sont rapportés dans le tableau 4.1.

Laquelle des expériences présentées dans l'article reproduit au chapitre 3 est la plus similaire à celle d'Ashton ?

- Au niveau du nombre de messages total, l'expérience rapportée à la section 3.4.3 avec un taux de 32 $^{msg}/_{s}$ est la plus près. Le nombre de paquets utilisés pour la synchronisation était de 14295. On constate un pourcentage de messages trop rapides de 6.44%.
- L'expérience ayant la durée la plus similaire est celle rapportée à la section 3.4.4. Celle-ci a une durée de 480 s et on y constate 6.01% de messages trop rapides.

On remarque un faible nombre d'inversions de message dans l'expérience d'Ashton malgré l'utilisation de l'algorithme basé sur les enveloppes convexes, alors que celui-ci garantit l'absence d'inversions. Cela est dû au fait qu'Ashton utilise la variante de l'algorithme décrite à la section 3.4.5. Cette variante permet d'obtenir des facteurs (au coût de quelques inversions) même lorsqu'il est impossible de faire passer une droite entre les deux groupes de points de la figure 3.2. Dans notre cas, l'algorithme de base s'est complété sans problème. On remarque d'autre part que le nombre de messages trop rapides est systématiquement plus faible dans notre cas. Cette différence pourrait s'expliquer par une latence plus faible au niveau du traceur et du réseau. Un autre article du même auteur révèle que le système utilise un traceur noyau et un réseau Ethernet (Ashton, 1995b). Il s'agit donc d'une configuration similaire à la nôtre. La performance du traceur utilisé n'est pas étudiée, mais, vu l'année de publication (1995), il est certain que le réseau utilisé était de catégorie Fast Ethernet (également introduit en 1995) (IEEE Std 802.3u, 1995) dans le meilleur des cas. Or, les expériences des sections 3.4.3 et 3.4.4 ont été réalisées sur un réseau Gigabit Ethernet. La section 3.4.2 a montré une amélioration significative (de 12.8% à 6.9%) en passant de l'un à l'autre. Il est donc plausible que cette différence contribue à ce que le nombre de messages trop rapides soit plus faible dans notre cas que dans celui d'Ashton.

Tableau 4.1 Métriques moyennes d'évaluation de la synchronisation, traces MUSBUS. Extrait de Ashton (1995a)

Nombre de messages total	Durée (s)	Messages trop rapides	Inversions de message
133400.33	611.89	9.36%	0.07

4.4 Considérations logicielles

Lors de la revue de littérature, il a été identifié que les algorithmes développés en recherche ne possédaient souvent aucune implantation connue, que les implantations disponibles ne pouvaient traiter qu'un seul type de trace et que les outils de traçage utilisaient pour la plupart des algorithmes rudimentaires. Ces critiques ont été prises en compte lors de l'implantation de l'algorithme décrit au chapitre 3. Un cadre d'application (*framework*) a été conçu de façon à permettre son inclusion dans plusieurs outils, la synchronisation de plusieurs types de traces à partir de plusieurs types d'événements et avec plusieurs algorithmes. Le tout a été fait de façon modulaire afin de pouvoir agencer chacune des combinaisons à chacune des étapes à volonté. Ce cadre d'application (illustré à la figure 4.1) est déjà utilisé dans deux outils, supporte deux formats de trace, deux types d'événements et trois algorithmes. Un effort particulier a été fait pour intégrer ce code au visualiseur LTTV (associé au projet LTTng) et le rendre disponible à la communauté, en réponse à quoi des usagers extérieurs se sont manifestés.

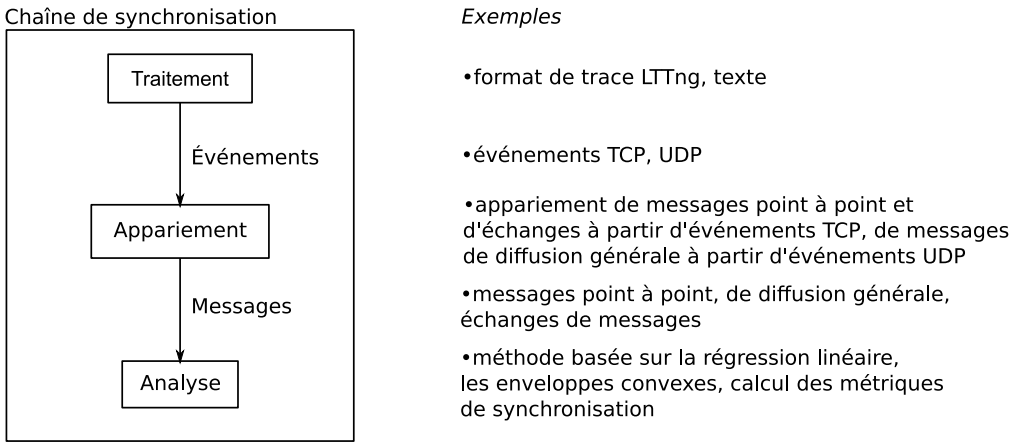


Figure 4.1 Architecture logicielle

CHAPITRE 5

CONCLUSION

L'algorithme présenté dans ce mémoire permet la synchronisation fidèle et juste de traces d'exécution enregistrées sur des systèmes répartis. Il s'agit d'une extension de l'algorithme basé sur les enveloppes convexes. Cette extension comporte la même précision que l'algorithme de base, mais permet en plus de déterminer des bornes sur la justesse de la synchronisation en tout point durant la période de traçage. Cet algorithme fournit une garantie sur l'absence d'inversion de message et une garantie de précision.

Nous avons décrit comment utiliser un traceur noyau pour enregistrer les événements réseau nécessaires à la synchronisation. L'endroit où enregistrer ces événements avec une faible intrusivité mais une grande applicabilité fut décrit. L'information minimale à enregistrer de façon à pouvoir apparier ces événements pour former des messages fut également décrite. Ceci donne une façon pratique de tracer n'importe quel programme échangeant des paquets TCP, sur tout type de matériel exécutant le noyau Linux, sans modification au programme et avec un impact minimal sur la performance du système.

Deux façons d'implanter l'algorithme basé sur les enveloppes convexes avec un ordre d'exécution linéaire en fonction du nombre de messages tracés furent décrites. Une première façon utilise le parcours de Graham pour construire les enveloppes convexes et un algorithme spécialisé pour identifier les facteurs nécessaires à la synchronisation. Une deuxième façon spécifie un programme linéaire pouvant identifier directement les facteurs à partir des messages ou après avoir trouvé les enveloppes convexes. L'implantation de l'extension permettant de déterminer les bornes sur la justesse dérive de ce programme linéaire. Le temps d'exécution de l'algorithme de base implanté selon la première façon, et le temps d'exécution de l'algorithme avec extension, furent mesurés expérimentalement sur plusieurs groupes de traces. Celles-ci ont été enregistrées sur des systèmes réels échangeant jusqu'à 18 000 000 d'événements. La version de base se comporte bel et bien de façon linéaire. La version étendue, bien qu'ayant un ordre quadratique dans le pire des cas, se comporte de façon quasi linéaire sur des traces réelles.

La précision de l'algorithme fut estimée dans différentes situations à l'aide de métriques basées sur les échanges de messages, les délais apparents lors de la réception de messages de diffusion générale ainsi que la justesse moyenne. La justesse de la synchronisation bénéficie de l'utilisation d'un réseau avec une latence plus faible ainsi que d'une augmentation du débit de messages. Au contraire, une durée de traçage plus longue diminue la précision et

fausse les bornes sur la justesse. Ceci fut détecté à l'aide de métriques basées sur les délais de transmission de messages point à point et de diffusion générale. Lors de nos expériences, nous avons atteint une justesse de ± 15 us et une précision estimée de 9 us sur un réseau comportant une latence minimale estimée de 39 us. Nous avons donc trouvé un algorithme de synchronisation de trace précis, qui garantit l'absence d'inversion de message en temps linéaire et qui peut déterminer des bornes sur la justesse dans un temps presque linéaire sur des traces réelles.

Le présent mémoire s'est concentré sur la synchronisation de paires de traces. Un sujet de recherche futur serait l'étude de la synchronisation de groupes de traces plus nombreuses. Il est déjà possible d'effectuer la synchronisation de tels groupes en synchronisant les traces deux à deux, en analysant les connexions entre les noeuds puis en propageant les facteurs. De telles techniques ne tiennent toutefois pas compte des bornes sur la justesse. Ces techniques devraient donc être raffinées, sans quoi les garanties de justesse sont perdues. De plus, les algorithmes qui s'assurent d'obtenir la meilleure précision nécessitent un temps d'exécution d'ordre quadratique en fonction du nombre de noeuds. Des approches alternatives, basées par exemple sur les messages de diffusion générale, pourraient être envisagées.

Un autre sujet qui mérite attention est la synchronisation de traces de longue durée. Plusieurs sources prônent la séparation des traces en sous-intervalles durant lesquels l'horloge est modélisée de façon linéaire. Aucune implantation d'algorithme effectuant ceci n'est toutefois connue. Il est probable que la mise en pratique de cette technique soulève de nouveaux aspects de recherche, notamment quant à la façon de joindre les sous-intervalles. L'évaluation de facteurs pour des fonctions de correction linéaires, mais l'utilisation de fonctions non linéaires telles des splines pourrait être envisagée.

Un troisième domaine d'intérêt est celui de l'utilisation des enveloppes convexes pour la synchronisation en ligne. Ceci est aussi lié à la synchronisation de traces en continu (*streaming*). Bien qu'à première vue ces deux sujets puissent sembler distincts de la synchronisation de traces de longue durée, ils s'en rapprochent sur certains points. La séparation des traces en sous-intervalles est une possibilité qui est également à étudier pour la synchronisation en ligne. Aborder un tel domaine brouillerait les frontières entre la synchronisation hors ligne et en ligne.

RÉFÉRENCES

ASHTON, P. (1995a). Algorithms for off-line clock synchronisation. Rapport technique, University of Canterbury, Department of Computer Science.

ASHTON, P. (1995b). An interaction network monitor for amoeba. Rapport technique, University of Canterbury, Department of Computer Science.

BECKER, D., RABENSEIFNER, R., WOLF, F. et LINFORD, J. C. (2009). Scalable timestamp synchronization for event traces of message-passing applications. 35, 595 – 607. Selected papers from the 14th European PVM/MPI Users Group Meeting.

BIBERSTEIN, M., HAREL, Y. et HEILPER, A. (2008). Clock Synchronization in Cell BE Traces. *Euro-Par '08 : Proceedings of the 14th international Euro-Par conference on Parallel Processing*. Springer-Verlag, Berlin, Heidelberg, 3–12.

BLIGH, M., DESNOYERS, M. et SCHULTZ, R. (2007). Linux kernel debugging on google-sized clusters. *Proceedings of the Linux Symposium*.

BROWNE, S., DONGARRA, J. et LONDON, K. (1998). Review of Performance analysis tools for MPI Parallel Programs. *NHSE Review*.

CENTER FOR INFORMATION SERVICES AND HIGH PERFORMANCE COMPUTING (ZIH) (2008). *VampirTrace 5.6.3 User Manual*. TU Dresden, Dresden, Germany.

CLÉMENT, E. (2006). *Synchronisation de traces dans un réseau distribué*. Mémoire de maîtrise, École Polytechnique de montréal.

CLÉMENT, E. et DAGENAIS, M. (2009). Traces synchronization in distributed networks. *Journal of Computer Systems, Networks, and Communications*, 2009.

CORRELL, K., BARENDT, N. et BRANICKY, M. (2005). Design considerations for software only implementations of the IEEE 1588 precision time protocol. *In Conference on IEEE 1588 Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*. vol. 1588.

COTEUS, P., BICKFORD, H., CIPOLLA, T., CRUMLEY, P., GARA, A., HALL, S., KOPCSAY, G., LANZETTA, A., MOK, L., RAND, R. *ET AL.* (2005). Packaging the blue gene/l supercomputer. *IBM Journal of Research and Development*, 49, 213–248.

CRISTIAN, F. (1989). A probabilistic approach to distributed clock synchronization. *Distributed Computing Systems, 1989., 9th International Conference on*, 288–296.

DEPARTMENT OF COMPUTER AND INFORMATION SCIENCE - UNIVERSITY OF OREGON (2009). Tuning and analysis utilities 2.18.2p2.

DESNOYERS, M. (2009). *Low-Impact Operating System Tracing*. Thèse de doctorat, École Polytechnique de Montréal.

DOLESCHAL, J., KNÜPFER, A., MÜLLER, M. S. et NAGEL, W. E. (2008). Internal timer synchronization for parallel event tracing. *Proceedings of the 15th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*. Springer-Verlag, Berlin, Heidelberg, 202–209.

DUDA, A., HARRUS, G., HADDAD, Y. et BERNARD, G. (1987). Estimating global time in distributed systems. *Proc. 7th Int. Conf. on Distributed Computing Systems, Berlin*. vol. 18.

EIDSON, J., FISCHER, M. et WHITE, J. (2002). IEEE-1588 standard for a precision clock synchronization protocol for networked measurement and control systems. *34 th Annual Precise Time and Time Interval (PTTI) Meeting*. 243–254.

ELLINGSON, C. et KULPINSKI, R. (1973). Dissemination of system time. *Communications, IEEE Transactions on*, 21, 605–624.

GWT-TUD GMBH (2008). Vampir news. Dresden, Germany.

HADDAD, Y. (1988). *Performance dans les systèmes répartis : des outils pour les mesures*. Mémoire de maîtrise, Université de Paris-Sud, Centre d'Orsay.

HOFMANN, R. et HILGERS, U. (1998). Theory and tool for estimating global time in parallel anddistributed systems. *Parallel and Distributed Processing, 1998. PDP'98. Proceedings of the Sixth Euromicro Workshop on*. 173–179.

IEEE STD 802.3U (1995). IEEE Standards for Local and Metropolitan Area Networks : Supplement to Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications Media Access Control (MAC) Parameters, Physical Layer, Medium Attachment Units, and Repeater for 100 Mb/s Operation, Type 100BASE-T (Clauses 21-30). *IEEE Std 802.3u-1995 (Supplement to ISO/IEC 8802-3 : 1993; ANSI/IEEE Std 802.3, 1993 Edition)*, 0_1–398.

JESKE, D. (2005). On maximum-likelihood estimation of clock offset. *IEEE Transactions on Communications*, 53, 53–54.

JEZEQUEL, J. (1989). Building a global time on parallel machines. *Proceedings of the 3rd International Workshop on Distributed Algorithms, LNCS*. Springer, vol. 392, 136–147.

JEZEQUEL, J. et JARD, C. (1996). Building a global clock for observing computations in distributed memory parallel computers. *Concurrency : Practice and Experience*, 8.

KNUPFER, A., BRUNST, H., DOLESCHAL, J., JURENZ, M., LIEBER, M., MICKLER, H., MULLER, M. et NAGEL, W. (2008). The Vampir Performance Analysis Tool-Set. *2nd HLRS Parallel Tools Workshop*. Springer.

LAMPORT, L. (1978). Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21, 558–565.

LANDES, T. (2007). Tree clocks : an efficient and entirely dynamic logical time system. *Proceedings of the 25th IASTED International Multi-Conference : parallel and distributed computing and networks*. ACTA Press Anaheim, CA, USA, 375–380.

MAKHORIN, A. (2009). GLPK (GNU Linear Programming Kit). *Free Software Foundation*.

MAROUANI, H. et DAGENAIS, M. (2008). Internal clock drift estimation in computer clusters. *Journal of Computer Systems, Networks, and Communications*, 2008, 1–7.

MATHEMATICS AND COMPUTER SCIENCE DIVISION - ARGONNE NATIONAL LABORATORY (2009). Mpich2 release 1.1.

MCCANNE, S. et JACOBSON, V. (1993). The BSD packet filter : A new architecture for user-level packet capture. *Proceedings of the 1993 Winter USENIX Technical Conference*. USENIX.

MESSAGE PASSING INTERFACE FORUM (1995). MPI : A Message-Passing Interface Standard.

MILLS, D. (1994). Precision synchronization of computer network clocks. *ACM SIGCOMM Computer Communication Review*, 24, 28–43.

MILLS, D. (2006). *Computer network time synchronization : the network time protocol*. CRC.

MILLS, D. et KAMP, P. (2000). The nanokernel. *Proc. Precision Time and Time Interval (PTTI) Applications and Planning Meeting*. Citeseer, 423–430.

NATARAJ, A., MALONY, A., SHENDE, S. et MORRIS, A. (2008). Integrated parallel performance views. *Cluster Computing*, 11, 57–73.

PRASAD, R., JAIN, M. et DOVROLIS, C. (2004). Effects of interrupt coalescence on network measurements. *Lecture Notes in Computer Science*, 247–256.

PROEBSTEL, E. (2008). *Characterizing and Improving Distributed Network-based Intrusion Detection Systems (NIDS) : Timestamp Synchronization and Sampled Traffic*. Mémoire de maîtrise, University of California Davis.

ROUSSEEUW, P. J. et DRIESSEN, K. (2006). Computing lts regression for large data sets. *Data Min. Knowl. Discov.*, 12, 29–45.

SCHEUERMANN, B. et KIESS, W. (2009). Who said that ? : the send-receive correlation problem in network log analysis. *ACM SIGMETRICS Performance Evaluation Review*, 37, 3–5.

SCHEUERMANN, B., KIESS, W., ROOS, M., JARRE, F. et MAUVE, M. (2009). On the time synchronization of distributed log files in networks with local broadcast media. *Networking, IEEE/ACM Transactions on*, 17, 431–444.

SIRDEY, R. et MAURICE, F. (2008). A linear programming approach to highly precise clock synchronization over a packet network. *4OR : A Quarterly Journal of Operations Research*, 6, 393–401.

TECHNISCHE UNIVERSITÄT DRESDEN - ZENTRUM FÜR INFORMATIONSDIENSTE UND HOCHLEISTUNGSRECHNEN (2009). VampirTrace version 5.6.3.

WOLF, F., WYLIE, B., ABRAHAM, E., BECKER, D., FRINGS, W., FÜRLINGER, K., GEIMER, M., HERMANNS, M., MOHR, B., MOORE, S. et SZEBENYI, Z. (2008). Usage of the SCALASCA Toolset for Scalable Performance Analysis of Large-Scale Parallel Applications. *Proceedings of the 2nd HLRS Parallel Tools Workshop*. 157–167.